The Dissertation Committee for Yuan Qu Certifies that this is the approved version of the following dissertation:

PICKUP AND DELIVERY PROBLEMS WITH SIDE CONSTRAINTS

Committee:

Jonathan F. Bard, Supervisor

Leon Lasdon

David P. Morton

Erhan Kutanoglu

Julian Pachon

PICKUP AND DELIVERY PROBLEMS WITH SIDE CONSTRAINTS

By

Yuan Qu, B.S.; M.A.; M.S.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

In Partial Fulfillment

of the Requirements

for the degree of

DOCTOR OF PHILOSOPHY

The University of Texas at Austin

December 2012

Dedication

This dissertation is dedicated to my family.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my mentor and supervisor Dr. Jonathan F. Bard for his constant guidance, understanding and encouragement during the entire course of this study. The scientific research of a doctoral degree is a lengthy process, especially for a part time student like me. I would not have been able to complete this study without Dr. Bard's patience and support which helped me get through all the difficult times.

I would also like to thank all my committee members: Dr. Leon Lasdon, Dr. David Morton, Dr. Erhan Kutanoglu and Dr. Julian Pachon, for their valuable advice on my research and critical reading of this dissertation.

I am very grateful to my colleagues at Caleb Technologies Corp. and Navitaire Inc, including Dr. Mark Song and Dr. Ben Thengvall for introducing me to the field of Operations research; Dr. Mike Arguello for his encouragement; Dr. Gang Yu for his recommendation; Marc Anderson, Dr. Saaju Paulose and Kevin Gibson for their support and allowing me to work around my class times. Special thanks also goes to Chris Deck at via Christi HOPE Center for providing the data and assisting me in refining the requirements for the daily planning problem at the PACE organization.

I would like to especially thank my parents, my in-laws and my sister for always being the source of love, support, and encouragement to me.

Lastly, the persons to whom I am deeply indebted are my wife Hui, my daughter Jessamine, and my son Michael. Both of my children were born in the years of my Ph.D. study. They have sacrificed a lot of quality family time to support my work on this dissertation. Their love, patience, understanding, and encouragement make my life so enjoyable and are indispensable to this dissertation.

PICKUP AND DELIVERY PROBLEMS WITH SIDE CONSTRAINTS

Yuan Qu, Ph.D.

The University of Texas at Austin, 2012

Supervisor: Jonathan F. Bard

Pickup and delivery problems (PDPs) have been studied extensively in past decades. A wide variety of research exits on both exact algorithms and heuristics for generic variations of the problem as well as real-life applications, which continue to spark new challenges and open up new opportunities for researchers. In this dissertation, we study two variations of pickup and delivery problem that arise in industry and develop new computational methods that are shown to be effective with respect to existing algorithms and scheduling procedures found in practice.

The first problem is the pickup and delivery problem with transshipment (PDPT). The work presented here was inspired by a daily route planning problem at a regional air carrier. In structuring the analysis, we describe a unique way to model the transshipment option on a directed graph. With the graph as the foundation, we implemented a branch and price algorithm. Preliminary results showed that it has difficulty in solving large instances. As an alternative, we developed a greedy randomized adaptive search procedure (GRASP) with several novel features. In the construction phase, shipment requests are inserted into routes until all demand is satisfied or no feasible insertion exists. In the improvement phase, an adaptive large neighborhood search algorithm is used to reconstruct portions of the feasible routes. Specialized removal and insertion heuristics were designed for this purpose. We also developed a procedure for generating problem instances in the absence of any in the literature. Testing was done on existing PDP data sets and generated PDPT data set. For the former, the performance and solution quality of the GRASP

were comparable to the best known heuristics. For the latter, GRASP found the near optimal solution in most test cases.

In the second part of the dissertation, we focus on a new version of the heterogeneous PDP in which the capacity of each vehicle can be modified by reconfiguring its interior to satisfy different types of customer demands. The work was motivated by a daily route planning problem arising at a senior activity center. A fleet of configurable vans is available each day to transport participants to and from the center as well as to secondary facilities for rehabilitative and medical treatment. To find solutions, we developed a two-phase heuristic that makes use of ideas from greedy randomized adaptive search procedures with multiple starts. In phase I, a set of good feasible solutions is constructed using a series of randomized procedures. A representative subset of those solutions is selected as candidates for improvement by solving a max diversity problem. In phase II, an adaptive large neighborhood search (ALNS) heuristic is used to find local optima by reconstructing portions of the feasible routes. Also, a specialized route feasibility check with vehicle type reassignment is introduced to take full advantage of the heterogeneous nature of vehicles. The effectiveness of the proposed methodology is demonstrated by comparing the solutions it provided for the equivalent of several weeks with those that were used in practice and derived manually. The analysis indicates that anywhere from 30% to 40% savings can be achieved with the multi-start ALNS heuristic.

An exact method is introduced based on branch and price and cut for settings with more restricted time windows. In the procedure, the master problem at each node in the search tree is solved by column generation to find a lower bound. To improve the bound, subset-row inequalities are applied to the variables of the master problem. Columns are generated by solving the pricing subproblems with a labeling algorithm enhanced by new dominance conditions. Local search on the columns is used to quickly find promising alternatives. Implementation details and ways to improve the performance of the overall procedure are discussed. Testing was done on a set of real instances as well as a set of randomly generated instances with up to 50 customer requests. The results show that optimal solutions are obtained in majority of cases.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

Pickup and delivery problems (PDPs) are at the center of a variety of applications in the transportation industry such as cargo delivery (Xu et al. 2003), home healthcare (Cordeau and Laporte 2003), and driver work area design (Bard and Jarrah 2009). In its basic form, the objective is to construct a set of minimum cost routes to satisfy transportation requests subject to flow balance, customer demand, vehicle capacity, and time window constraints. The problem can be viewed as a vehicle routing problem (VRP) with precedence constraints, in which each pickup has to be performed before the delivery for a given request.

There are many variants of PDP with regard to vehicle type, request type, and request handling. Demand can be serviced by either a homogeneous fleet or a heterogeneous fleet of vehicles with different capacities and operational costs. In the latter case, we have a heterogeneous PDP (HPDP) (Xu et al. 2003). When there is more than one request type, each may have different capacity requirements, and hence can only be serviced by the vehicles that possess the required capacity (Parragh 2011). In the vast majority of cases, the capacity of a vehicle is fixed but in our second study, we consider it to be adjustable or configurable. If a single request can be split into several smaller units and handled by multiple vehicles, then we have the case of a *split load* (Desaulniers 2010). If a single request can be transferred from one vehicle another at some intermediary locations, then we have a *transshipment* option (Shang and Cuff 1996).

In general, the objective is to minimize a weighted sum consisting of one or more of the following: the total number of vehicles used (Shang and Cuff 1996, Nanry and Barnes 2000, Li and Lim 2001, Xu et al. 2003, Ropke and Pisinger 2006); the total distance traveled by the vehicles (Shang and Cuff 1996, Li and Lim 2001, Xu et al. 2003, Ropke and Pisinger 2006); the number of requests that are not satisfied (Shang and Cuff 1996, Nanry and Barnes 2000); the total time vehicles are operational (Li and Lim 2001); vehicle waiting time (Shang and Cuff 1996, Li and Lim 2001, Xu et al. 2003) and shipment or passenger ride time (Parragh 2011). Regardless of the specific form, the PDP is NP-hard in the strong sense since it is a generalization of the VRP.

In this dissertation we investigate two variants of the PDP inspired by real-life applications. The first concerns daily planning at regional cargo carrier, where shipments are picked up and delivered by aircraft. The objective is to satisfy customer demands while minimizing the overall operational costs that mainly include maintenance and fuel costs. We model it as a pickup and delivery problem with transshipment (PDPT) and propose an exact method based on column generation. Due to poor performance, a GRASP was developed to solve instances comparable to those arising in practice. The second application centers on a daily route planning problem arising at a senior activity center. A fleet of configurable vans is available each day to transport participants to and from the center as well as to secondary facilities for rehabilitative and medical treatment. The number of participants and support equipment that a van can accommodate depends on how it is configured. We model the problem as a heterogeneous PDP with configurable vehicle capacity (HPDPCC). The management at the center wants to evaluate the solutions under different settings. Based on the problem characteristics under different settings, we developed a multi-start adaptive large neighborhood search heuristic and an exact solution method based on branch and price and cut. Accordingly, the following sections of the dissertation are organized into two parts: PDPT and HPDPCC. In Chapter 2, we focus on the PDPT. We first introduce the problem and then provide a literature review in Section 2.2. In Section 2.3, we give a formal definition of the problem and illustrate it with a small example. In Section 2.4, we present a mix integer formulation of the problem. In Section 2.5, we first introduce a branch and price solution method, followed by description of the GRASP and explain the test case generation process. In Section 2.6, the performance of the branch and price method on generated PDPT data instances are discussed. Using six well known PDP data sets the performance of the GRASP is analyzed and shown to be at least as effective as existing heuristics. We also tested our procedure on five newly generated PDPT data sets with 10 instances each. Solutions that are within 1% optimality were obtained 88% of the time. Conclusions are drawn in Section 2.7 where some plans for future research are outlined.

Chapter 3 investigates the HPDPCC beginning with an introduction and a literature review in Section 3.1 and 3.2, respectively. A detailed description of problem characteristics is given in Section 3.3. In Section 3.4, we give a formal definition of the problem and present a mixed integer formulation. In Section 3.5, we introduced two solution methods, a multi-start

adaptive variable neighborhood search and a branch and price and cut algorithm. In Section 3.6, the computation results for both methods are presented. Conclusions and future research ideas are given in Section 3.7.

Chapter 4 provides a summary of the solution methods proposed in this dissertation along with future research directions.

Chapter 2. Pickup and Delivery Problem with Transshipment

2.1 INTRODUCTION

The pickup and delivery problem (PDP) has wide application in the transportation industry with the most prominent examples being express mail delivery (Bard and Jarrah 2009), dial-a-ride (Cordeau and Laporte 2003) and cargo transport (Xu et al. 2003). In its basic form, the problem is to construct a set of minimum cost routes to satisfy transportation requests subject to a variety of constraints. In our work, these include a fixed number of identical vehicles each with limited capacity, a single depot where each route starts and ends, an origin and destination for each request with a hard pickup and delivery time window, and perhaps restrictions on the duration of a route. Furthermore, for each shipment there is a service time that indicates how long it takes to perform the pickup and delivery operation. If a vehicle arrives at a location early, it must wait until the start of the time window before initiating service.

There are several variations of this problem with regard to the handling of a request by more than one vehicle (Shang and Cuff 1996, Thangiah et al. 2007, Desaulniers 2010). If a single shipment can be split into several smaller shipments and handled by multiple vehicles, then we have the case of a *split load*. If a single shipment can be transferred from one vehicle to other vehicles at some intermediary locations, then we say *transshipment* is allowed. Given that the number of vehicles is limited, we might encounter situations where some requests cannot be routed.

In this chapter, the focus is on the pickup and delivery problem with transshipment (PDPT). This interest was triggered by a real application associated with daily route planning for a regional air cargo carrier. The general problem that these carriers face can be described as follows (Crainic and Laporte 1997, Yan et al. 2006). For a given service area and set of customers, requests come in daily for goods to be transported from some origin O to some destination D. In most cases, the O-D pairs are airports that have curfews that cannot be violated. The goods are usually delivered to the airports by third party logistics providers and ground transport companies. Each shipment is marked with an "earliest available time" and "latest delivery times." The carriers have to determine the resources such as aircraft, airports and transshipment locations required to handle the demand. In addition, there are legal and technique restrictions, such as crew working hours and aircraft maintenance schedules that have to be

considered when constructing a schedule. The objectives are to either minimize the number of vehicles required to meet demand or minimize the cost of providing service with the full fleet.

Accordingly, in this chapter we introduce a new graphical representation of the PDPT that captures the transshipment options, and present a MIP formulation. We implement a column generation based exact method which is suitable for small sized problems. We then describe a greedy randomized adaptive search procedure (GRASP) that we have developed for finding solutions (Boudia et al. 2006, Feo et al. 1991, Kontoravdis and Bard 1995). In the first phase of the GRASP, feasible schedules are efficiently constructed by sequentially inserting either customers or transshipment nodes in a route. In the improvement phase, we apply and extend the adaptive large neighborhood search (ALNS) methodology proposed by Shaw (1998) and Ropke and Pisinger (2006) to hierarchically minimize the required number of vehicles and then to minimize the distance traveled. We also introduce an innovative way for constructing test problems whose costs are minimized only when transshipments are included in the solution.

In the next section, we discuss the related literature. In Section 2.3, we give a formal definition of the problem and illustrate it with a small example. In section 2.4 we present the MIP formulation. In Section 2.5, we present two solution methods: branch and price and GRASP. In Section 2.6, the computation results are shown for branch and price and the GRASP respectively. Conclusions are drawn in Section 2.7 where some plans for future research are outlined.

2.2 LITERATURE REVIEW

The basic PDP has been extensively studied for the last two decades. One can find detailed reviews by Savelsbergh et al. (1995), Mitrović-Minić (1998) and Berbeglia et al. (2007). Both exact methods and heuristics have been applied. Optimization methods adapted for the problem typically involve some forms of column generation (Desrosiers and Dumas 1988, Dumas et al. 1991, Ropke and Cordeau 2009) or cutting plan techniques (Ruland and Rodin 1997, Ropke et al. 2007). Heuristics usually take a phased approach, and may include clustering, route construction and route improvement (Nanry and Barnes 2000, Li and Lim 2001, Ropke and Pisinger 2006).

There are many real-world applications that can be represented as a PDP. The dial-a-ride problem is a special case in which passengers are to be transported from one location to another. Besides time window constraints, maximum ride times and other quality-of-service related

restrictions may be included. Cordeau and Laporte (2003) survey solution methods. The express mail and small package delivery problem is another example that has been widely investigated. Barnhart et al. (2002) and Armacost et al. (2002) have published several papers on service network design for express shipment delivery by air carriers. Their research has mainly been at the strategic and tactic planning levels, where the objectives are to determine what routes a carrier should cover, how frequently the routes should be served, and what paths should be followed. They applied their models and algorithmic techniques to solve a network design problem at UPS which operates under a standard hub-spoke system. In this setting, all shipments are picked up at origin airports and flown to hubs for sorting; sorted shipments are then delivered to the destination airports. Direct shipments from origin to destination airports are not permitted nor are mixtures of pickup and delivery routes––each type of service is scheduled separately. To ensure the robustness of the schedules and that operations remain simple, at most three stops in a pickup route or in a delivery route were allowed, so each had at most two legs.

Kim et al. (1999) also worked on the transportation service network design problem developing a route-based model that captured operating regulations and policies as well as the complex cost relationships that exist in the express package environment. By applying a series of novel reduction methods, they achieved a dramatic decrease in problem size without compromising optimality. A specialized branch-and-price-and-cut based heuristic was used to find solutions. Improvements to the heuristic are discussed by Barnhart et al. (2002) who report a 16% reduction in total annual costs for the sponsoring company with the new design. Continuing this work, Armacost et al. (2002) took advantage of the problem characteristics and introduced the concept of composite variables. This helped them obtain strong lower bounds, which allowed the problem to be solved by traditional branch and bound.

A related but more routine problem concerns the pickup and delivery of regular mail. Because the number of customers is huge, it is common to divide the service area into zones and then to assign a vehicle to each. This simplifies the daily routing problem. Langevin and Soumis (1989) developed an approximate analytical method for zone planning. In a related effort Laporte et al. (1989) studied operations at Canada Post and implemented a clustering and routing algorithm to improve the efficiency and reduce the number of vehicles required for mailbox collections. The clustering algorithm first partitioned the customers into groups whose expected

demand did not exceed the capacity of a truck. Routes were then constructed by solving a generalized travel salesman problem.

In contrast, there have been only a few papers on the PDPT. Shang and Cuff (1996) proposed a look-ahead heuristic for picking up and delivering patient records, equipment and supplies for a health maintenance organization (HMO). The heuristic first constructs mini-routes based on shipment requests and then incorporates them into the vehicle routes. This is done with an insertion procedure aimed at finding the best work schedule, that is, the best vehicle in which to consolidate the relevant items and thus achieve the greatest efficiency. As with most such heuristics, the potential number of insertions can be enormous but due to the ready times, deadlines and precedence relations, it was possible to reduce the number of alternatives down to a manageable size. The performance of the heuristic was tested with real data provided by the HMO, which typically delivers (approximately) 300 items to nine locations over 13 hours each day using six vehicles. The heuristic provided a 37.4% savings on average over current practice.

Thangiah et al. (2007) extended the above idea to include a local search phase. Their overlap heuristic redistributes shipments from the vehicle that has the smallest number of shipments to vehicles that are more heavily loaded. This procedure helps to reduce the fleet size. Their exchange heuristic is similar to their overlap method except that it moves shipments between arbitrary vehicles to obtain new routing alternatives. The authors claimed that solution quality greatly improved with the local search.

Mitrović-Minić et al. (2006) proposed a standard two-phase heuristic to solve the PDPT allowing one transshipment per request. The construction phase consisted of a random multi-start greedy insertion procedure. Several solutions are derived using different random initial orderings of the requests, which are greedily inserted into existing partial routes one by one. The best solution found is used as the initial solution for the improvement phase in which the current solution is modified by successively removing and reinserting each request. We take a similar approach. To allow for maximum flexibility a request may be split during the construction phase and recombined during the improvement phase. Also, a request may be split and unsplit several times during the improvement phase. In each phase, a customized procedure is called that evaluates the insertion cost of an unsplit request and for each transshipment point, evaluates two insertions of a split pair of requests. The cheapest insertion is performed.

The authors randomly generated test cases with 50 or 100 requests whose locations were either uniformly distributed or concentrated in clusters. Their results showed that the use of transshipment points reduced the total distance traveled when the requests were uniformly generated, especially for the larger instances. For the clustered instances, the benefits increased as the clusters got smaller.

Cortés et al. (2009) proposed a branch-and-cut algorithm that made use of combinatorial Benders cuts, a mechanism developed by Codato and Fischetti (2006) to overcome weak LP relaxations due to big-*M* values in mixed-integer programs (MIPs). They achieved better performance with their approach when compared to standard branch and bound for test cases with up to 6 requests. Yan et al. (2006) addressed a short-term planning problem for an air cargo carrier, and developed an integrated scheduling methodology that could be used to simultaneously determine aircraft routes, cargo flow paths and flight schedules. For a 7-day planning horizon, potential flight schedules were constructed under the assumption that flights depart from stations every 4 hours; that is, at 00:00, 04:00, 08:00 and so on. Their MIP formulation included a fleet-flow time-space network and a multi-cargo-flow time-space network. Several heuristics were proposed and their performance compared in a case study using operational data provided by a major Taiwan airline. The main idea is to solve the MIP directly while limiting the number of arcs on which a shipment is permitted to flow from its origin to destination. They start with a "nonstop" network (only one flight leg per request) and find a solution. The number of permissible legs is then incrementally increased for those requests with large origin-to-destination distances, and the problem is re-solved. The process is repeated until no improvement is realized.

In summary, current heuristics provide efficient ways to obtain good solutions to the PDP but there have been only a handful methods proposed for the PDPT and little can be said about their performance. Moreover, no standard data sets are available for testing new algorithms.

2.3 PROBLEM DEFINITION

The PDPT can be formally described on a directed graph $G = (N, A)$. The node set $N = P \cup T \cup \{b, b'\}$, where P is the node set associated with customer requests, T is the node set associated with transshipment locations and b and b' are the depot start node and end node. The arc set A consists of all feasible links, including dummy links within the set T, as described below.

Each customer request $c \in C$ is associated with a load requirement L_c to be picked up at node i_c and delivered to node j_c. All vehicles $k \in K$ are stationed at the depot where they start and end their day, servicing customers along the way. The sequence of nodes visited by a vehicle including the depot nodes b and b' is called a route $r \in R$. For all $i \in N$, a time window $[a_i, b_i]$ is defined that specifies the earliest and latest time service can begin, regardless of node designation. If a vehicle arrives at i prior to a_i, it has to wait until a_i to start service, which has a fixed duration of σ_i hours.

The travel time T_{ij} between two nodes i and j is determined by the vehicle speed V_k and distance D_{ij} between the locations associated with the nodes: $T_{ij} = D_{ij}/V_k$. A customer request $c \in C$ can be handled by one or more vehicles as they traverse their routes. When a single vehicle provides the service, it visits the pickup and delivery nodes of the request. When multiple vehicles provide service, the request may be picked up at either node $i_c \in P$ or one of its transshipment nodes $i_{ct} \in T$ at a transshipment location t, and it may be delivered to either node $j_c \in P$ or one of its transshipment delivery node $j_{ct} \in T$. If a request $c \in C$ goes through a transshipment node, then an additional service time is incurred at that node. The capacity Q_k of vehicle $k \in K$ cannot be exceeded along its route.

The first objective is to minimize the number of vehicles required to service all customer demand subject to capacity and time window constraints. When multiple solutions exist, the second objective is to minimize the total distance traveled, which is proportional to the variable cost of running the business. An underlying assumption is that transshipment does not incur additional loading and unloading costs, just additional time. This is the case in our motivating application where facilities, equipment and labor costs are part of the business infrastructure and so are constant whether or not transshipment is used. All loading and unloading activities are performed by full-time workers whose pay does not depend on the type of service required at a particular location. Nevertheless, our solution methodology described in the next section can be easily extended to include transshipment handling costs in the objective function.

The node structure of the network G is defined as follows. For each customer request $c \in C$, there is one pickup node i_c and one delivery node j_c, and at each transshipment location t, there is one transshipment pickup node i_{ct} and one transshipment delivery node j_{ct} again for each customer c. As mentioned, there is also the single depot start node b and end node b'. For

example, a problem with two customer requests c_1 and c_2, and one transshipment location t, gives rise to a total of 10 nodes as shown in Figure 2.1. In addition to b and b', nodes i_{c_1} and j_{c_1} are pickup and delivery nodes for customer c_1, i_{c_2} and j_{c_2} are pickup and delivery nodes for customer c_2, i_{c_1t} and j_{c_1t} are the transshipment delivery and pickup nodes for the customer c_1 at the transshipment location t; and i_{c_2t} and j_{c_2t} are the transshipment delivery and pickup nodes for customer c_2 at the transshipment location t.

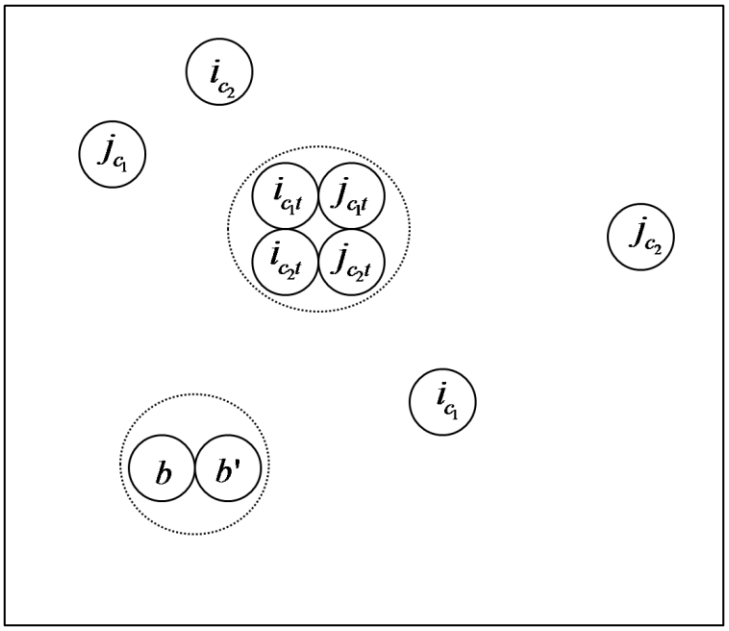

Figure 2.1. Node structure for PDPT

Proposition 2.1. There exists an optimal solution with the following property. If a vehicle visits a request node or a transshipment node, then it either picks up or drops off the shipment for the request. All nodes except depot nodes are visited at most once.

Proof. If an optimal solution contains a vehicle route that visits a request or transshipment pickup node but does not pick up the shipment, than this node can be removed from the route. Based on triangle inequality, the new route would traverse the same or shorter distance. Similarly if a route visits a request or transshipment delivery node but it does not deliver the shipment, the node does not need to be included in the route. If a node is visited more than once, that means a customer request is picked up and delivered more than once. In such a situation, we can keep one

pair of pickup and delivery nodes in the route and remove the others while still satisfying the demand. The new route would be feasible and have the same or shorter travel distance. ■

According to *Proposition* 2.1, each node $i \in N\backslash\{b, b'\}$ is visited at most once, so we can associate with it a unique service start time t_i, load q_i on the vehicle just prior to the arrival, and time window $[a_i, b_i]$. Directed arcs exist between most nodes except in the following cases: there are no arcs from any node to b nor are there any arcs from b' to any other node; no customer or transshipment pickup node can connect to b' (all shipments must be delivered before the vehicle returns to the depot); no arc between b and customer or transshipment delivery nodes (no shipment can be delivered before being picked up); a customer or transshipment pickup node cannot join a transshipment pickup node of the same customer (if a shipment has been picked up, there is no need to pick it up again); and, no arc between a customer or transshipment delivery node and the customer pickup node or a transshipment pickup or delivery node of the same customer (no need to pick up or delivery a shipment right after it is dropped off). These arcs are either precluded by precedence restrictions or unnecessary in light of *Proposition* 2.1. Nevertheless, when using heuristics, precedence restrictions cannot be enforced by simply omitting arcs that would lead to a direct violation. During the construction phase, partial routes must be checked continually for violations over a sequence of nodes.

Example 2.1. The following example illustrates how to construct the graphical representation of a PDPT. Assume that there is one vehicle, two customer requests, c_1 and c_2, and one transshipment location t. All locations are defined on a Euclidean plane with Cartesian coordinates $<x, y>$. For customer c_1, a shipment of 8 units must be picked up at origin $<4,0>$ within the interval $[0,10]$ and dropped off at destination $<0,4>$ within the interval [5, 20]. For customer c_2, a shipment of 6 units must be picked up at $<3,1>$ within the interval $[2,10]$ and delivered to $<1,3>$ within the interval $[2,20]$. All service times σ_i, for $i \in N\backslash\{b, b'\}$, are 0.1. The depot b is located at $<0,0>$ and is open between 0 and 20. The transshipment location t is $<2,2>$, the capacity of the vehicle is 10, and its speed is 1.

The corresponding graph has a total of 10 nodes: $b, b', i_{c_1}, j_{c_1}, i_{c_2}, j_{c_2}, i_{c_1t}, j_{c_1t}, i_{c_2t}$ and j_{c_2t}, but to avoid clutter, we label them 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9, respectively. When the feasible arcs are included, the graph is fairly dense, but, as mentioned, there are several types of flow that are excluded. In particular, arcs entering node 0 and leaving node 1 are infeasible; no customer

or transshipment pickup node can connect to 1 [that is, arcs (2, 1), (4, 1), (7, 1), (9, 1) are disallowed]; no arc can go from 0 to customer or transshipment delivery nodes [that is, arcs (0, 3), (0, 5), (0, 6), (0, 8) are disallowed]; a customer or transshipment pickup node cannot connect to a transshipment pickup node of the same customer [that is, arcs (2, 7), (7, 2), (4, 9), (9, 4) are disallowed]; and no arc can go from a customer or transshipment delivery node to the customer pickup node or a transshipment pickup or delivery node of the same customer [that is, arcs (6, 2), (6, 3), (6, 7), (8, 4), (8, 5), (8, 9) are disallowed].

Table 2.1 provides detailed information for each node. Columns 2 and 3 identify the customers and their demand, respectively. Column 4 indicates whether the load is to be picked up (+) or delivered (–). Columns 5 - 7 give the node locations, service start time windows, and service time, respectively, and the last column lists the nodes to which the current node is not connected in the graph.

Table 2.1. Detailed node information

Node I	Customer c	Customer demand L_c	Pickup/delivery	Location (x, y)	Time window $[a_i, b_i]$	Service time σ_i	Nodes cannot be connected to
$b = 0$				<0,0>	[0, 20]	0	3, 5, 6, 8
$b' = 1$				<0,0>	[0, 20]	0	2, 3, 4, 5, 6, 7, 8, 9
$i_{c_1} = 2$	c_1	8	+	<4,0>	[0, 10]	0.1	0, 1, 7
$j_{c_1} = 3$	c_1	8	–	<0,4>	[5. 20]	0.1	0
$i_{c_2} = 4$	c_2	6	+	<3,1>	[2, 10]	0.1	0, 1, 9
$j_{c_2} = 5$	c_2	6	–	<1,3>	[2, 20]	0.1	0
$i_{c_1t} = 6$	c_1	8	+	<2,2>	[0, 20]	0.1	0, 2, 3, 7
$j_{c_1t} = 7$	c_1	8	–	<2,2>	[0, 20]	0.1	0, 1, 2
$i_{c_2t} = 8$	c_2	6	+	<2,2>	[0, 20]	0.1	0, 4, 5, 9
$j_{c_2t} = 9$	c_2	6	–	<2,2>	[0, 20]	0.1	0, 1, 4

Figure 2.2 depicts all 10 nodes and two solutions. The transshipment nodes 6, 7, 8, 9 are at the same physical location, as are the depot start and end nodes 0 and 1. The first solution (dashed arrows) is the route {0, 2, 3, 4, 5, 1} and does not make use of the transshipment node. The total

distance traveled is 18.47 and the total travel time is 18.87. If we allow transshipment, the optimal route (solid arrows) is {0, 2, 6, 4, 5, 7, 3, 1} with a total distance of 17.90 and a total travel time of 18.50. Here, the shipment is not actually transferred from one vehicle to another vehicle but is dropped off at node 6 and picked up by the same vehicle at node 7. That is, the vehicle arrives at node 2 to pick up the load for customer c_1 and temporarily drops it off at node 6. The empty vehicle then picks up the load for customer c_2 at node 4, delivers it to node 5, returns to the transshipment location, pickups up the waiting load for c_1 at node 7, and delivers it to node 3 before returning to the depot. In other scenarios, a shipment might be transferred from one vehicle to another to increase capacity on the first or to reduce travel times. ■

Example 2.1 exhibits several other characteristics of the PDPT. If the depot closing time is reduced to 18.6 so the service time window for node 1 is [0, 18.6], then the optimal solution without transshipment is no longer feasible unless a second vehicle is available. The best route with transshipment, though, is still valid. Also, the shortest route without transshipment is {0, 2, 4, 5, 3, 1} with a distance of 13.66 and a travel time of 14.06, but this route violates the capacity constraint. In Section 2.5.2, we give another example in which the number of vehicles and the total distance traveled can be reduced with transshipments. In general, transshipment locations can be used to temporarily store shipments while a vehicle serves nearby customers.

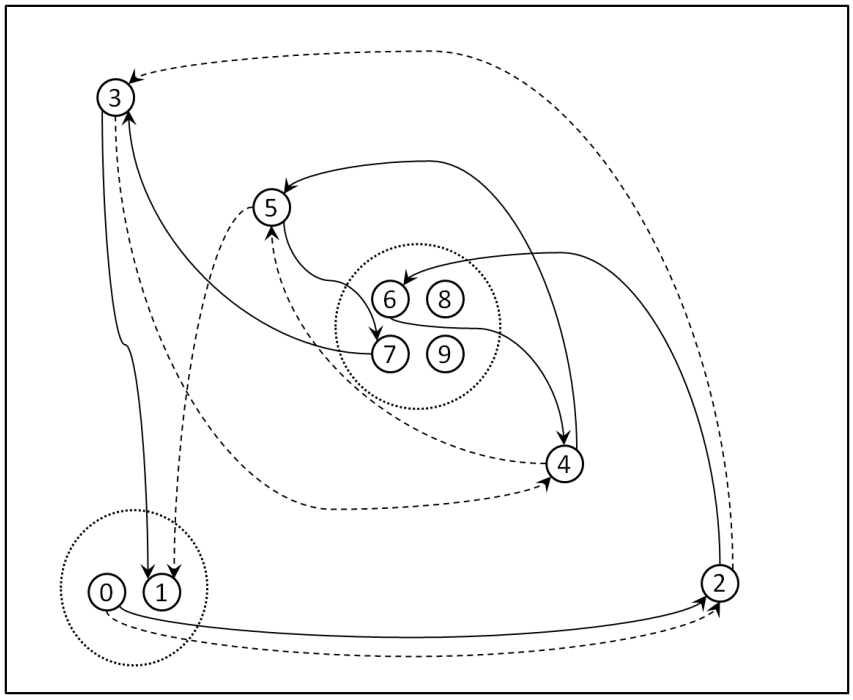

Figure 2.2. Network representation for Example 2.1

We now give two conditions under which transshipment can be beneficial.

Proposition 2.2. For the single vehicle PDPT, a necessary condition for a route with transshipment to provide a reduction travel distance is that the total customer demand exceeds the capacity of the vehicle.

Proof. In Example 2.1, we have shown that transshipment can reduce travel time and distance for a single vehicle PDPT. We now show that this is true only when the vehicle capacity is tight. Specifically, if there is no capacity restriction or the vehicle has sufficient capacity to handle all customer requests, then given a solution *S* for a single vehicle PDPT that visits one or more transshipment locations, we can construct a new solution $\bar{S}$ by removing all transshipment nodes from the route in *S*. Because the vehicle capacity is not tight, the new solution $\bar{S}$ is feasible, and due to the triangle inequality, would traverse either the same or a shorter distance than the original solution. Finally if there were time windows for a subset of the requests, removing the transshipment nodes from *S* would not produce a time window violation for $\bar{S}$ since the vehicle is permitted to wait at a customer location prior to beginning service. ■

Proposition 2.3. In a PDPT with two or more vehicles, transshipment can be beneficial even when the vehicle capacity is not tight.

Proof. To begin, construct an instance Ψ of a PDPT that has a unique optimal solution S with two routes r_1 and r_2 that intersect at location t. Assuming r_1 arrives at t before r_2, we can construct a new instance $\bar{\Psi}$ by adding a new customer c toΨ. Let the pickup location for c be on r_1's trajectory before location t and the delivery location for c be on r_2's trajectory after location t. Assume that the load for c is small enough and can be inserted into either r_1 or r_2 at any point, and without transshipment let the optimal solution of $\bar{\Psi}$ be $\bar{S}$. Because S is unique and $\bar{\Psi}$ includes an additional customer, we know that the total distance for $\bar{S}$ is larger than that of S; otherwise, we could remove the pickup and delivery nodes for c from routes in $\bar{S}$ to get a solution for Ψ that is better than S. Now, if we use location t as a transshipment point, we can assign the pickup node for customer c to r_1 and the delivery node to r_2 to create a new solution. As a result, the total distance traveled in this new solution for $\bar{\Psi}$ is identical to S and hence less than $\bar{S}$. Thus, the use of transshipment can reduce the total distance traveled. ■

Note that *Proposition* 2.3 is applicable even when transshipment handling costs are included in the problem. In the proof, an instance Ψ can always be constructed with large distances between locations. This would marginalize the transshipment handling costs and make them irrelevant.

2.4 MODEL

In this section, we present an arc-flow formulation for the PDPT based on the directed graph $G = (N, A)$ described in previous section. The indexes and sets for the formulation are defined as the following:

Indices and sets

k	index for vehicles
K	a set of vehicles
c	index for customer requests
C	a set of customer requests
i, j	indices for nodes
N	set of nodes including base nodes, pickup nodes, delivery nodes, transshipment nodes
b, b'	indices for base start node and end node
PN	set of pickup nodes

DN set of delivery nodes

i_c pickup node index for request $c \in C$

j_c delivery node index for request $c \in C$

TD set of transshipment drop off nodes

TP set of transshipment pickup nodes

$TD(c)$ set of transshipment drop off nodes for request $c \in C$

$TP(c)$ set of transshipment pickup nodes for request $c \in C$

$TD(c)$ set of delivery nodes

t_c ,t_c' indices for transshipment pickup node and delivery node for request $c \in C$ at a transshipment location

A set of arcs

TA set of arcs between $t_c \in TD$ and corresponding $t_c' \in TP$ for all customer requests and transshipment locations

$TA(c)$ set of arcs between $t_c \in TD$ and corresponding $t_c' \in TP$ for request $c \in C$ and all transshipment locations

$NF(i)$ set of nodes that node i has arc to connect to

$NB(i)$ set of nodes that have arcs to connect to node i

Parameters

c_{ij} routing cost for arc $(i, j) \in A$

T_{ij} travel time for arc $(i, j) \in A$, service time are included

L_c shipment load for customer $c \in C$

Q vehicle capacity

(a_i, b_i) time window for node $i \in N$

Decision variables

x_{ij}^k binary variable that is 1 when vehicle k serves arc (i, j), 0 otherwise

y_{ij}^{ck} binary variable that is 1 when shipment for customer c is on vehicle k from i to j, 0 otherwise

t_i service start time at node i

$$\text{Minimize} \sum_{k \in K} \sum_{(i,j) \in A \setminus TA} c_{ij} x_{ij}^k \tag{2.1a}$$

subject to

Vehicle flow balance constraints

$$\sum_{j \in NF(b)} x_{bj}^k = 1, \forall k \in K \tag{2.1b}$$

$$\sum_{j \in NB(i)} x_{ji}^k - \sum_{j \in NF(i)} x_{ij}^k = 0, \forall i \in N, k \in K \tag{2.1c}$$

$$\sum_{i \in NB(b')} x_{ib'}^k = 1, \forall k \in K \tag{2.1d}$$

Shipment flow balance constraints

$$\sum_{k \in K} \sum_{j \in NF(i_c)} y_{i_c j}^{ck} = 1, \forall c \in C \tag{2.1e}$$

$$\sum_{j \in NB(i)} y_{ji}^{ck} - \sum_{j \in NF(i)} y_{ij}^{ck} = 0, \forall c \in C, i \in N \setminus (TP(c) \cup TD(c) \cup \{i_c, j_c\}), k \in K \tag{2.1f}$$

$$\sum_{k \in K} \sum_{i \in NB(j_c)} y_{ij_c}^{ck} = 1, \forall c \in C \tag{2.1g}$$

$$\sum_{k \in K} \sum_{j \in NF(t_c)} y_{jt_c}^{ck} - \sum_{k \in K} \sum_{j \in NF(t_c')} y_{t_c' j}^{ck} = 0, \forall c \in C, (t_c, t_c') \in TA(c) \tag{2.1h}$$

Vehicle capacity constraints

$$\sum_{c \in C} L_c y_{ij}^{ck} \le Q x_{ij}^k, \forall k \in K, (i,j) \in A \setminus TA \tag{2.1i}$$

Time continuity constraints

$$t_i + T_{ij} \le t_j + (1 - \sum_{k \in K} x_{ij}^k) M, \forall (i,j) \in A \setminus TA \tag{2.1j}$$

$$t_i + T_{ij} \le t_j + (2 - \sum_{k \in K} \sum_{l \in NB(i)} x_{li}^k - \sum_{k \in K} \sum_{l \in NF(j)} x_{jl}^k) M, \forall (i,j) \in TA \tag{2.1k}$$

$$a_i \le t_i \le b_i, \forall i \in N \tag{2.1l}$$

Variable definitions

$$x_{ij}^k \in \{0,1\}, \forall k \in K, (i,j) \in A \setminus TA$$

$$y_{ij}^{ck} \in \{0,1\}, \forall k \in K, c \in C, (i,j) \in A$$

$$t_i \ge 0, \forall i \in N \tag{2.1m}$$

Valid inequalities

$$y_{ij}^{ck} = x_{ij}^{k}, \forall k \in K, \forall c \in C, \forall j \in N, \forall i \in TP(c) \cup \{i_c\} \quad (2.1n)$$

$$y_{ij}^{ck} = x_{ij}^{k}, \forall k \in K, \forall c \in C, \forall i \in N, \forall j \in TD(c) \cup \{j_c\} \quad (2.1o)$$

$$\sum_{k \in K} \sum_{i \in NF(i)} x_{ij}^{k} \leq 1, \forall j \in N \setminus \{b, b'\} \quad (2.1p)$$

$$y_{ij}^{ck} \leq x_{ij}^{k}, \forall k \in K, \forall c \in C, \forall (i, j) \in A \setminus TA \quad (2.1q)$$

$$t_i + T_{ij} \leq t_j + (1 - x_{ij}^{k})M, \forall k \in K, \forall (i, j) \in A \setminus TA \quad (2.1r)$$

The objective function (2.1a) minimizes the total routing costs which consist of arc costs travelled by vehicles. For an arc <*i*, *j*>, mostly the cost c_{ij} is the total travel cost between nodes except for cases where *i* is depot start node and *j* is any node except depot end node. In those cases, c_{ij} is the total travel cost between nodes plus the vehicle operating cost. Constraints (2.1b) and (2.1d) ensure that the route of each vehicle *k* starts and ends at the base depot, specifically, supply of 1 is given at each vehicle start node by constraints (2.1b) and demand of 1 is given at each vehicle end node ensured by constraints (2.1d). Constraints (2.1c) ensure that vehicle flow is balanced at customer and transshipment nodes. Constraints (2.1e)-(2.1f) guarantee that flow is balanced for each customer request. Constraints (2.1e) and (2.1g) ensure that supply of 1 and demand of 1 are given at each customer request's pickup and delivery node respectively. Constraints (2.1f) ensure that shipment flow for a customer request is balanced on a vehicle at each node except the pickup and delivery nodes for the same customer. The flow balance between a transshipment delivery node and its pick node is satisfied by constrains (2.1h). The vehicle capacity restriction is ensured by constraints (2.1i). Constrains (2.1j) and (2.1k) ensure that the time continuity is satisfied at the two nodes connected by an arc, with (2.1k) for any arc connecting two transshipment pickup and delivery nodes for the same customer at the same transshipment location, (2.1j) for rest of arcs. Constraints (2.1*l*) ensure that visiting time at a node satisfies time window restrictions. Variable definitions are given in (2.1m). Some valid inequalities are given in (2.1n)-(2.1r) to strengthen the formulation. Based on *Proposition* 2.1, if a route is to visit a customer or transshipment pickup node, it has to pick up the shipment. This is guaranteed by constraints (2.1n). Similarly constraints (2.1o) ensure that a route delivers the shipment if it visits a customer or transshipment delivery node. Constraints (2.1p) are to ensure

that all nodes except depot nodes are visited at most once. Constraints (2.1q) can be derived from (2.1i) while (2.1r) can be derived from (2.1j).

2.5 SOLUTION METHODS

Certain instances of pickup and delivery problems with time windows can reliably be solved with specialized optimization codes for up to 50 customer requests, and less frequently with several hundred customers (e.g., see Ropke and Cordeau 2009). When the transshipment option is included in the formulation, the problem becomes much more difficult. We first present a branch and price exact algorithm to solve the problem. The initial computation results show that it can only be used for small sized instances. As an alternative, we developed a GRASP using ALNS in phase II to improve the solutions found in phase I.

2.5.1 Branch and price

To solve the mixed integer problem presented in model (2.1), a branch and price method is used, where the lower bound at each node is computed by a column generation algorithm. The following is the main algorithm.

Procedure `Branch_and_Bound`

Input the graph $G=(N, A)$, vehicle data, shipment data

Output integral solution S

Step 0: Initialization

Get initial incumbent primal solution $S_{incumbent}$ use the GRASP;

Create root node for branch and bound BBN_{root} and initialize BBN_{root} with empty branch condition set $SET_{bc.}$;

Add BBN_{root} to the priority queue Q;

Step 1: Check if there are any nodes to be updated

If (Q is not empty)

Retrieve the first node *BBN* from Q and remove it from Q;

Solve node *BBN* by calling S_{lp} = `Column_Generation` (*BBN*);

If (S_{lp} is not feasible) go to Step 1;

Else if ($S_{lp} \geq S_{incumbent}$) go to Step 1;

Else

If (S_{lp} is integral)

If ($S_{incumbent} > S_{lp}$) $S_{incumbent} = S_{lp}$;

Else

Identify branch object (arc to branch on) using branching rules;

Create child nodes based on the branch object and update their $SET_{bc.}$;

Add the child nodes to Q;

Go to Step 1;

Step 2: Return $S_{incumbent}$;

Stop;

Figure 2.3. Pseudocode for branch and bound algorithm

The traversal of the branch and bound tree is implemented based on a priority queue. Either depth-first or best-first traversal can be easily implemented with this data structure which can accommodate different ways of defining priorities at the nodes. For depth-first traversal, we define the priority of a node using its depth in the branch and bound tree, the deeper it's in the tree, the higher its priority is. For best-first traversal, we define the priority of a node based on the linear programming (LP) relaxation solution of its parent node –– the better the LP bound, the

higher its priority. In the following sections, we will discuss the column generation algorithm and the branching rules in details.

Dantzig-Wolfe decomposition

To apply the Dantzig-Wolfe decomposition principle to model (2.1), one needs to identify the constraints that define the feasible domain of the subproblems. Let (2.1b)-(2.1d), (2.1f), (2.1i), (2.1n), (2.1o), (2.1q), (2.1r) be those constraints and $D = \{(x, y):(x, y)$ *satisfies the constraints* (2.1b)-(2.1d), (2.1f), (2.1i), (2.1n), (2.1o), (2.1q), (2.1r)$\}$ be the feasible domain of the subproblem. This bounded domain is separable by vehicle $k \in K$ and yields identical feasible bounded domain $D^k = \{(x^k, y^k):(x^k, y^k)$ *satisfies the constraints* (2.1b)-(2.1d), (2.1f), (2.1i), (2.1n), (2.1o), (2.1q), (2.1r)$\}$. A subproblem can be associated with each of these domains. Given these subproblems, the application of the Dantzig-Wolfe decomposition principle to model (2.1) corresponds to substituting (x^k, y^k) in model (2.1) except the constraints used to define the subproblem by a convex combination of the extreme points of D^k for all $k \in K$. This substitution yields the master problem. All subproblems can be aggregated into a single subproblem because they are identical. Due to *Proposition* 2.1, once a route that is a sequence of nodes visited is determined, the corresponding (x^k, y^k) in the subproblem is also determined. We present the master problem in terms of routes determined by the subproblem.

Master Problem

In formulating the master problem one modification we make with respect to the original model is to allow unsatisfied customer requests. In particular, we introduce the variable u_c to account for an unsatisfied customer request and a cost c_c to penalize it in the objective function. This helps us construct an initial solution in the column generation algorithm to construct.

Indices and sets

r	index of routes
R	a set of routes

Parameters

c_r	cost of route $r \in R$
c_c	cost of unsatisfied customer $c \in C$
a_{ir}	binary parameter that is 1 if node i is visited by route r

b_{ijr} binary parameter that is 1 if arc (i, j) is visited by route r

Decision variables

z_r binary variable that is 1 when route r is used

u_c binary variable that is 1 when customer c is not served

$$\text{Minimize} \sum_{r \in R} c_r z_r + \sum_{c \in C} c_c u_c \tag{2.2a}$$

subject to

$$\sum_{r \in R} a_{i_c r} z_r + u_c = 1, \forall c \in C \tag{2.2b}$$

$$\sum_{r \in R} a_{j_c r} z_r + u_c = 1, \forall c \in C \tag{2.2c}$$

$$\sum_{r \in R} a_{ir} z_r - \sum_{r \in R} a_{jr} z_r = 0, \forall (i, j) \in TA \tag{2.2d}$$

$$t_i + T_{ij} \le t_j + (1 - \sum_{r \in R} b_{ijr} z_r) M, \forall (i, j) \in A / TA \tag{2.2e}$$

$$t_i + T_{ij} \le t_j + (2 - \sum_{r \in R} a_{ir} z_r - \sum_{r \in R} a_{jr} z_r) M, \forall (i, j) \in TA \tag{2.2f}$$

$$a_i \le t_i \le b_i, \forall i \in N \tag{2.2g}$$

$$\sum_{r \in R} a_{ir} z_r \le 1, \forall i \in N / \{b, b'\} \tag{2.2h}$$

$$\sum_{r \in R} z_r \le | K | \tag{2.2i}$$

$$z_r \in \{0,1\}, \forall r \in R$$

$$t_i \ge 0, \forall i \in N \tag{2.2j}$$

The objective function (2.2a) is to minimize total routing cost. Constraints (2.2b)-(2.2h) are equivalent to constraints (2.1e), (2.1g), (2.1h), (2.1j), (2.1k), (2.1*l*) and (2.1p). Constraint (2.2i) is the convex constraint.

Subproblem

Consider the following dual variables:

π_c dual variables of constraints (2.2b) for customer $c \in C$

β_c dual variables of constraints (2.2c) for customer $c \in C$

α_{ij} dual variables of constraints (2.2d) for arc $(i, j) \in TA$

γ_{ij} dual variables of constraints (2.2e) for arc $(i, j) \in A\backslash TA$

δ_{ij} dual variables of constraints (2.2f) for arc $(i, j) \in TA$

θ_i nonnegative dual variables of constraints (2.2h) for node $i \in N\backslash\{b, b'\}$

λ dual variable of constraint (2.2i)

The reduce cost in terms of a solution of subproblem (x, y) can be written as

$$\sum_{(i,j)\in A/TA} c_{ij}x_{ij} - \sum_{j\in N/\{b,b'\}} \theta_j \sum_{i\in N} x_{ij} - \sum_{c\in C} \pi_c \sum_{j\in N} y^c_{i_c j} - \sum_{(i,j)\in TA} \alpha_{ij}\left(\sum_{l\in N} x_{li} - \sum_{l\in N} x_{jl}\right) - \sum_{c\in C} \beta_c \sum_{i\in N} y^c_{ij_c}$$

$$- \sum_{(i,j)\in A/TA} \gamma_{ij} M x_{ij} - \sum_{(i,j)\in TA} \delta_{ij} M\left(\sum_{l\in N} x_{li} + \sum_{l\in N} x_{jl}\right) - \lambda$$

Now, we define $c^1_{ij} = \begin{cases} c_{ij} - \gamma_{ij} M & \forall (i,j) \in A \setminus TA \\ 0 & otherwise \end{cases}$, $c^2_{ij} = \begin{cases} \theta_j & \forall i \in N, j \in N \setminus \{b, b'\} \\ 0 & otherwise \end{cases}$,

$c^3_{ij} = \begin{cases} -\sum_{(j,l)\in TA} (\delta_{jl} M + \alpha_{jl}) & \forall i \in N, j \in TD \\ 0 & otherwise \end{cases}$, $c^4_{ij} = \begin{cases} -\sum_{(l,i)\in TA} (\delta_{li} M - \alpha_{li}) & \forall i \in N, j \in TP \\ 0 & otherwise \end{cases}$ and

$\tilde{c}_{ij} = c^1_{ij} + c^2_{ij} + c^3_{ij} + c^4_{ij}$ for coefficients of x_{ij} in the reduce cost. Also define

$d^{c1}_{ij} = \begin{cases} \pi_c & \forall c \in C, i = i_c \\ 0 & otherwise \end{cases}$, $d^{c2}_{ij} = \begin{cases} \beta_c & \forall c \in C, i = j_c \\ 0 & otherwise \end{cases}$ and $d^c_{ij} = d^{c1}_{ij} + d^{c2}_{ij}$.

Consequently, the subproblem can be represented as the following.

Decision variables

x_{ij} binary variable that is 1 when route service arc (i, j)

y^c_{ij} binary variable that is 1 when shipment for customer c is on arc (i, j)

$$\text{Minimize} \sum_{(i,j)\in A/TA} \tilde{c}_{ij} x_{ij} - \sum_{c\in C} \sum_{(i,j)\in A} d^c_{ij} y^c_{ij} - \lambda \qquad (2.3a)$$

subject to

$$\sum_{j\in NF(j)} x_{bj} = 1 \qquad (2.3b)$$

$$\sum_{j \in NF(i)} x_{ji} - \sum_{j \in NT(i)} x_{ij} = 0, \forall i \in N \setminus \{b, b'\} \quad (2.3c)$$

$$\sum_{i \in NF(b')} x_{ib'} = 1 \quad (2.3d)$$

$$\sum_{j \in NB(i)} y_{ji}^c - \sum_{j \in NF(i)} y_{ij}^c = 0, \forall c \in C, i \in N \setminus (TP(c) \cup TD(c) \cup \{i_c, j_c\}) \quad (2.3e)$$

$$\sum_{c \in C} q_c y_{ij}^c \leq Q x_{ij}, \forall (i, j) \in A \setminus TA \quad (2.3f)$$

$$t_i + T_{ij} \leq t_j + (1 - x_{ij}) M, \forall (i, j) \in A \setminus TA \quad (2.3g)$$

$$a_i \leq t_i \leq b_i, \forall i \in N \quad (2.3h)$$

$$y_{ij}^c = x_{ij}, \forall c \in C, (i, j) \in A, i \in TP(c) \cup \{i_c\} \quad (2.3i)$$

$$y_{ij}^c = x_{ij}, \forall c \in C, (i, j) \in A, j \in TD(c) \cup \{j_c\} \quad (2.3j)$$

$$y_{ij}^c \leq x_{ij}, \forall c \in C, \forall (i, j) \in A \setminus TA \quad (2.3k)$$

$$x_{ij} \in \{0,1\}, \forall (i, j) \in A \setminus TA$$

$$y_{ij}^c \in \{0,1\}, \forall c \in C, (i, j) \in A$$

$$t_i \geq 0, \forall i \in N \quad (2.3l)$$

The objective of the subproblem defined by model (2.3) is to find a route with minimum reduced cost as in (2.3a). Constraints (2.3b)-(2.3k) are equivalent to constraints (2.1b)-(2.1d), (2.1f), (2.1i), (2.1r), (2.1*l*), (2.1n), (2.1o) and (2.1q), respectively, but decomposed by vehicle $k \in K$.

Column Generation

The relaxed master problem defined by model (2.2) is solved by column generation while columns or routes are generated by solving the pricing subproblem defined by model (2.3). By introducing another variable u_c for unsatisfied request, we don't need initial columns for the master problem, but the large cost associated with unsatisfied demand leads to large dual values making the subproblem harder to solve, so it's beneficial to start the column generation with good initial columns. In the following, we present the column generation algorithm.

Procedure `Column_Generation`

Input the graph $G=(N, A)$, vehicle data, shipment data

Output solution S_{LP} specifies vehicle route assignments.

Step 0: Initialization

Build restricted master model *RMP* with empty routes;

Set new route set SET_r empty;

Find initial routes by heuristic (optional), add them to SET_r;

Step 1: Add routes in SET_r to *RMP*;

Solve *RMP*, set S_{LP} with the solution of *RMP*;

Get dual information *DUAL* of *RMP*;

Solve subproblem by calling SET_r = `Solve_Subproblem`(*DUAL*);

If (SET_r is not empty)

Go to Step 1;

Step 2: Return S_{LP};

Stop;

Figure 2.4. Pseudocode for column generation algorithm

The subproblem can be viewed as a resource constrained shortest path problem, which can be solved using dynamic programming techniques called labeling algorithms. Given a graph $G = (N, A)$ with a source node and a sink node, labeling algorithm builds partial paths in the graph, each partial path starts at the source node and ends at a node $i \in N$. The existing partial paths are extended along the arcs leaving the end node of the partial path by applying an extension function. A path is found if the end node is the sink node.

In principle all possible paths are enumerated in this algorithm; however, some partial paths at an end node may be dominated by other partial paths with the same end nodes, so these partial paths can be removed from the search. The major components of this algorithm are the definition of the label, the extension function, and the dominance condition. In our subproblem the graph is the same as previously defined in Section 2.3. The source node is the depot departure node while the sink node is the depot arrival node. A feasible partial path from the depot departure node to any node $i \in N$ is represented by a label L associated with node i. $L = \{P, C, T, W, RP, RD, \{RP_t\}, \{RD_t\}\}$ where P is a sequence of nodes visited by the partial path. C is the reduced cost

for the partial path. T is the earliest feasible service start time at node i. W is the total shipment on the last arc of the partial path. RP is a set of requests picked up including the ones delivered. RD is a set of requests being delivered to their destination. RD_t is a set of requests dropped at the transshipment location t. RD_t is a set of requests picked up at the transshipment location t. If a label for node i is $L_i = \{P_i, C_i, T_i, W_i, RP_i, RD_i, \{RP_{ti}\}, \{RD_{ti}\}\}$, we extend the label along arc (i, j) and the resulting label is $L_j = \{P_j, C_j, T_j, W_j, RP_j, RD_j, \{RP_{tj}\}, \{RD_{tj}\}\}$. The extension functions are defined as the following.

$$P_j = \{P_i, j\}$$

$$C_j = C_i + c_{ij} - \sum_{c \in \{c: i_c \in RP \cup \{RP_t\} / RD \cup \{RD_t\}\}} d_{ij}^c$$

$$T_j = \max(T_i + T_{ij}, a_j)$$

$$W_j = W_i + \sum_{c \in C,\, j = i_c \cup j \in TP(c)} L_c - \sum_{c \in C,\, j = j_c \cup j \in TD(c)} L_c$$

$$RP_j = RP_i \cup \{c : j = i_c\}$$

$$RD_j = RD_i \cup \{c : j = j_c\}$$

$$RP_{tj} = RP_{ti} \cup \{c : j = t_c'\}$$

$$RP_{tj} = RP_{ti} \cup \{c : j = t_c\}$$

If $T_j > b_j$ or $W_j > Q$ then the partial path represented by L_j is infeasible, can be removed. Given two labels $L^1 = \{P^1, C^1, T^1, W^1, RP^1, RD^1, \{RP_t^1\}, \{RD_t^1\}\}$ and $L^2 = \{P^2, C^2, T^2, W^2, RP^2, RD^2, \{RP_t^2\}, \{RD_t^2\}\}$, L^1 dominates L^2 if $C^1 \le C^2$, $T^1 \le T^2$, $W^1 \le W^2$, $RP^1 = RP^2$, $RD^1 = RD^2$, $\{RP_t^1\} = \{RP_t^2\}$ and $\{RD_t^1\} = \{RD_t^2\}$. The following is the pseudo code for the labeling algorithm.

Procedure `Solve_Subproblem`

Input Graph $G = (N, A)$ with source node b and sink node b'

Output Best path found L^*

Step 0: Initialization

Let $P^0 = \{b\}$, $C^0 = -\lambda$, $T^0 = a_b$, $W^0 = 0$, $RP^0 = \varnothing$, $RD^0 = \varnothing$ and $RP_t^0 = \varnothing$, $RP_t^0 = \varnothing$ for all $t \in T$;

Let $L^1 = \{P^0, C^0, T^0, W^0, RP^0, RD^0, \{RP_t^0\}, \{RD_t^0\}\}$;

Let $L^* = L^0$;

Initialize priority queue by putting $PQ \leftarrow L^0$;

Initialize node label association structure $\{LS_i, i \in N\backslash\{b'\}\}$, where LS_i is the set of labels associated with node i and initially are all empty;

Step 1: If $(PQ = \varnothing)$

Remove label L from top of the queue PQ, set i as the last node visited by L;

For each arc (i,j) extended from node i,

Get label $L_j = \{RC_j, UN_j, C_j, T_j, VL_j\}$by applying the extension function to L along (i,j);

If L_j is feasible

Check if L_j is dominated by labels in LS_j;

If L_j is not dominated, then

put $PQ \leftarrow PQ \cup \{L_j\}$;

put $LS_j \leftarrow LS_j \cup \{L_j\}$;

If ($j = b'$ and $C_j < C^*$), then set $L^* = L_j$;

Go to Step 1.

Step 2: Return L^*;

Stop.

Figure 2.5. Pseudocode for labeling algorithm

One variation to the algorithm is to allow multiple paths to be returned and early termination of the algorithm. All paths with negative reduce cost will potentially help improve the restricted

master problem. The algorithm can be terminated as long as one path with negative reduce cost is found.

Initial primal solution

We use a GRASP-based heuristic as described in the next section to create an initial primary solution.

Branching rules

An arc based branching scheme is applied in the algorithm. The branching rule is similar to the one proposed by Ryan and Foster (1981). If the solution of column generation is fractional, we try to identify an arc in routes such that $\sum_{r\in R} b_{ijr} z_r$ is fractional. Based on *Proposition* 2.1, we know that $\sum_{r\in R} b_{ijr} z_r \le 1, \forall < i, j > \in A \setminus < b, b' >$ because all nodes in *G* except depot nodes can be visited at most once. We know that in a fractional solution, we can always find such <*i*, *j*> that $\sum_{r\in R} b_{ijr} z_r$ is fractional. Because otherwise, all vehicle routings are fixed, we would have found an integer solution.

In our implementation, the arc with fractional $\sum_{r\in R} b_{ijr} z_r$ which is closest to 0.5 is selected, when there is a tie, it's broken arbitrarily. Once an arc is identified, two new child nodes will be created from their parent node in branch and bound tree. The routes in parent node that violates the branch decision will be discarded. The branch decision objects are carried in the child nodes, since a branch decision will potentially change master problem and subproblem. In our case, if an arc $(i, j) \in A$ is selected to branch on, in one branch x_{ij} is set to 1, the routes in the parent node which visit node *i* but not followed by node *j* are removed from the corresponding child node, the subproblem algorithm will be updated to ensure if any route visits node *i* will visit node *j* next; in the other branch x_{ij} is set to *0*, the routes in parent node that visit node *i* immediately followed by node *j* are removed from the child node, the subproblem algorithm will be updated to ensure that any route visiting node *i* will not visit node *j* next.

2.5.2 GRASP

Our computational experience with the exact branch and price method proposed in the previous section shows that the method is only suitable for small instances containing a few customer

requests. As an alternative, we developed a GRASP using ALNS in phase II to improve the solutions found in phase I.

In the context of metaheuristics, researchers have had notable success in solving large instances of VRPs using two types of local search algorithms. Simulated annealing (Li and Lim 2001) and tabu search (Nanry and Barnes 2000) represent the first type in which "small" neighborhoods are explored at each step in an effort to improve the current solution. Large neighborhood search (LNS) is the second type, and greatly expands the topological landscape. At each step, a substantial part of the current solution is destroyed and then reconstructed, thus providing a much broader search. When different types of neighborhoods are defined, we get what is often called *variable neighborhood search* (Hansen and Mladenovic 2001). Recently, Ropke and Pisinger (2006) used LNS on VRPs and PDPs with time windows to get high quality solutions to instances with up to 500 customers. They extended the idea proposed by Shaw (1998) to arrive at what they called ALNS. This approach begins with an initial solution and iteratively destroys and rebuilds it by randomly choosing and applying a number of quick neighborhood search heuristics. Associated with each heuristic is a weight that determines its selection probability. At each iteration, the new solution is either accepted or rejected and the heuristic selection weights are updated according to their performance. For the PDP, a solution is destroyed by removing a number of customers from the routes using tailor-made procedures, and then reinserting them back into perhaps different routes, depending on the logic of the insertion heuristics.

A flow diagram of our GRASP is outlined in Figure 2.6. Feasible solutions are constructed sequentially in Phase I using $|K|$ vehicles. We start with $|K|$ null routes such that each contains only the start and end depot nodes, and then individually insert requests into the routes until all shipments are assigned to vehicles or no more request can be accommodated due to capacity or time window restrictions. In Phase II, we apply ALNS in an attempt to improve the initial solution. The overall procedure is terminated after a predefined number of iterations. Before presenting the details, we need discuss the insertion operations which constitute the building blocks of the algorithms used in both phases.

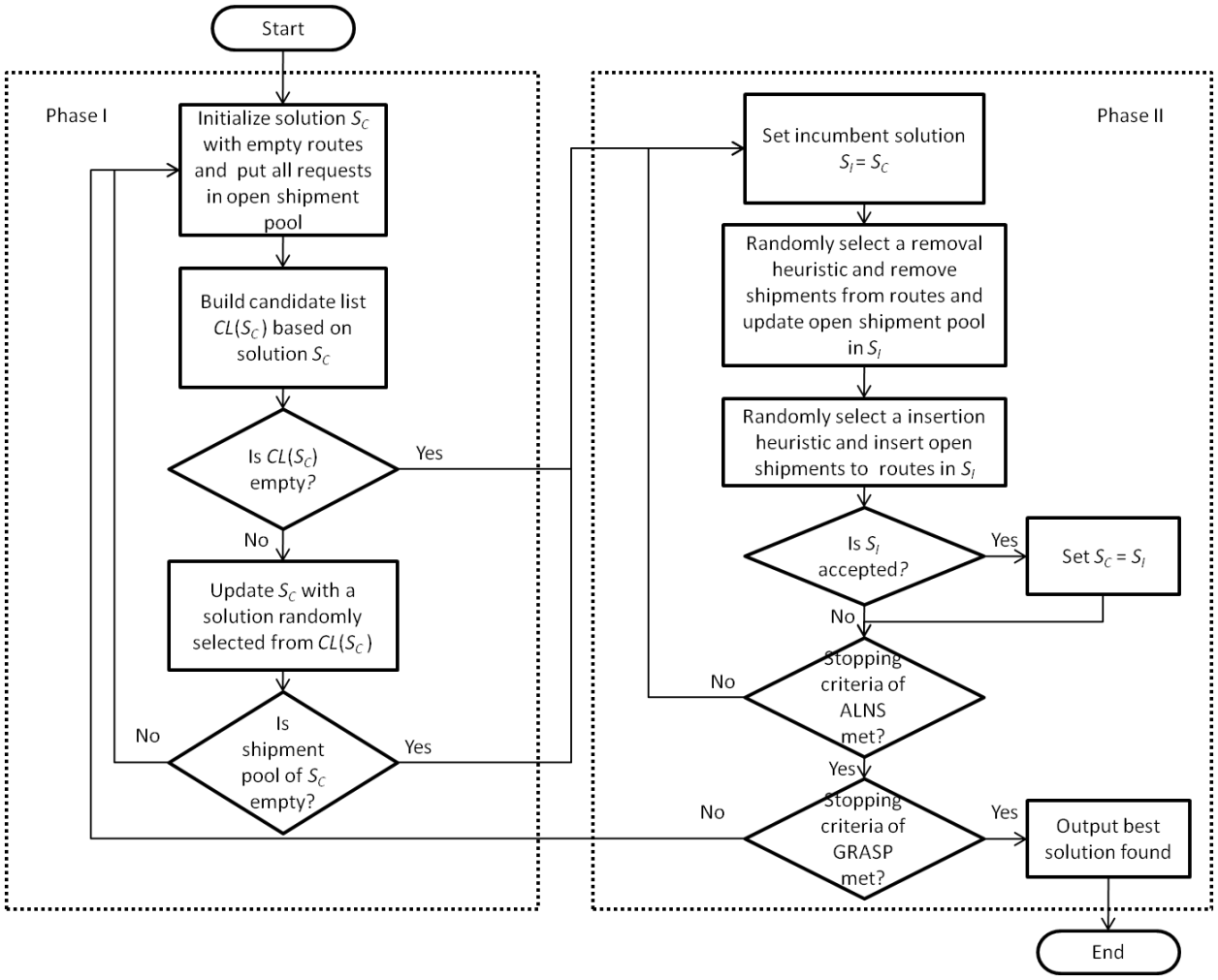

Figure 2.6. GRASP flow diagram

Basic insertion operation

Insertions are the basis for constructing feasible solutions to a wide variety of routing and scheduling problems. For the PDP, a pair of nodes *i* and *j* that are associated with the pickup and delivery locations of a customer are inserted into an existing route. For the PDPT, it was necessary to extend the logic to allow for the possibility of transshipments. The first step is to define the shipment request $i \rightarrow j$ for customer *c*, where the pickup node *i* can be either i_c or i_{ct} at transshipment location *t*. Similarly the delivery node *j* can be j_c or j_{ct}. Depending on whether a transshipment location is used or not, there are two types of basic insertion operations. When no transshipment is involved we use *single route insertion* where a request $i \rightarrow j$ is inserted into an existing route r_1. This is shown in Figure 2.7. Node *i* can be inserted into any position in the

route r_1 after the depot start node b and before the depot end node b'. Node j can be inserted into any position after node i but before b'.

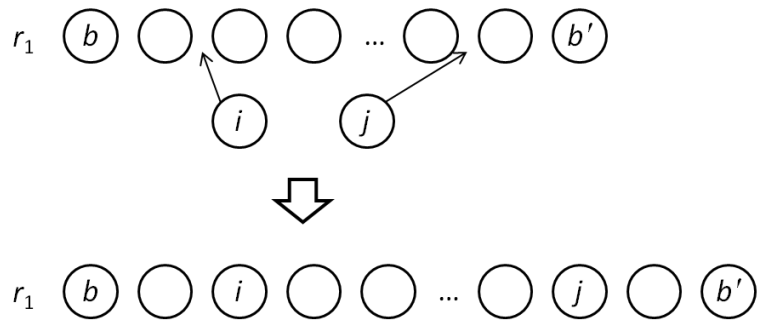

Figure 2.7. Single route insertion: request $i \rightarrow j$ is inserted into route r_1

When transshipment is being considered, we use *double route insertion*, where a request $i \rightarrow j$ for customer c is inserted into two routes r_1 and r_2 (that are not necessarily distinct). For transshipment location t and customer c a pair of transshipment nodes j_{ct} and i_{ct} is used to split the shipment request into two requests $i \rightarrow j_{ct}$ and $i_{ct} \rightarrow j$. The former $i \rightarrow j_{ct}$ is inserted into route r_1 using *single route insertion*; the latter $i_{ct} \rightarrow j$ is inserted into route r_2 using *double route insertion*. This is illustrated in Figure 2.8 for distinct routes. When r_1 and r_2 are the same, i_{ct} is inserted after j_{ct} but they must be separated by at least one customer node to preclude suboptimal transitions. This case is illustrated in Figure 2.9. After a change is made to a route, the service start time of each customer on the route as well as the load on the vehicle must be updated, and the feasibility of the updated route checked.

Feasibility check and solution update

When transshipments are not permitted, the path of each shipment follows the path of one vehicle so only one route is affected by an insertion (or removal) operation. In the case of the PDPT, a shipment path can potentially overlap several vehicle paths, implying that the route update and feasibility check are more complicated. We developed a propagation algorithm to perform these steps.

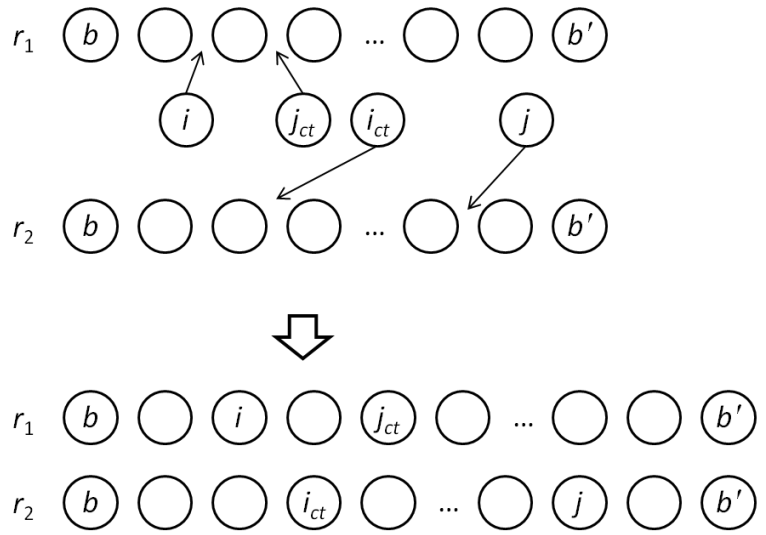

Figure 2.8. Double route insertion: request $i \rightarrow j$ is inserted into routes r_1 and r_2 using transshipment nodes j_{ct} and i_{ct}

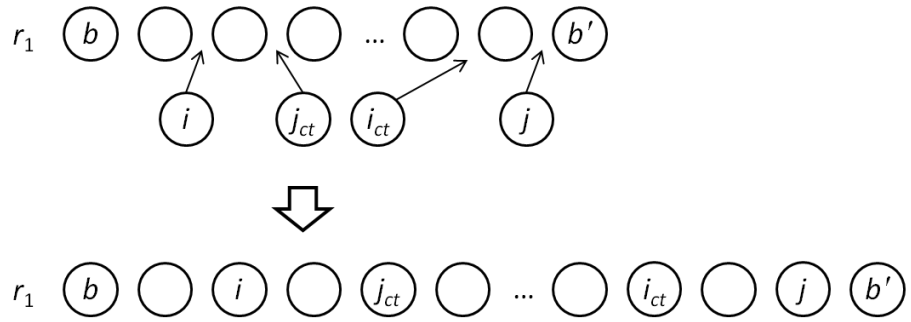

Figure 2.9. Double route insertion: request $i \rightarrow j$ is inserted into route r_1 using transshipment nodes j_{ct} and i_{ct}

A (partial) solution S of a PDPT contains the following information: (1) a set of routes R, where route $r \in R$ is represented by a sequence of nodes that starts at b and ends at b'. If r only contains b and b', then the route is said to be null or empty; (2) for each node i in a route, the service start time t_i and the load on the vehicle q_i just prior to visiting i; and (3) the unsatisfied shipment requests U, where a shipment request $i \rightarrow j \in U$ if the pair of nodes has not been served by any of the $|K|$ vehicles. When a solution S is modified, i.e., a shipment request is either inserted into a possibly new route or removed from an existing route, the solution information must be updated accordingly. In the case of an insertion, we need update the affected routes in R

by including nodes i and j, and removing $i \rightarrow j$ from U. In the case of a removal, i and j are removed from the affected routes in R, and $i \rightarrow j$ is added to U.

Each time a request $i \rightarrow j$ is added to or removed from R, the `Check_Feasibility` procedure outlined in Figure 2.10 is used to update service start times t_i and t_j, and vehicle loads q_i and q_j in all affected routes and to check for feasibility with respect to the service start time windows and vehicle capacity constraints. The check starts with an initial node set NL which is determined as follows. If request $i \rightarrow j$ is inserted using *single route insertion*, we add nodes i and j to NL. In the case of *double route insertion* using transshipment nodes t and t', nodes i, j, t and t' are added to NL. If request $i \rightarrow j$ is removed from the current solution, we add the nodes right after i and j in their original routes to NL. The following example illustrates how the set NL is constructed.

Example 2.2. Given an original route $r = \{b, i_1, i_2, i_3, \ldots, i_{k-1}, i_{k+1}, \ldots, i_n, b'\}$, if i_2 and i_{k-1} are the nodes to be removed, we need add i_3 and i_k to NL because that's where the updating should start. Alternatively, if i_k is inserted into route r, we need to add i_k to NL because the service time and load information for each of its successors must be updated. ■

Once the affected nodes are identified, the current solution S and the set NL are input to `Check_Feasibility`, which returns either the updated service start time and load information or a flag indicating that the modification is infeasible. In the worst case, the algorithm loops through all the nodes in $N \setminus \{b,b'\}$. Since the computational effort for each node is $O(1)$, the time complexity of the full procedure is $O(|N|)$.

Procedure `Check_Feasiblity`
Input : Current solution *S*, set of nodes *NL* that are affected by modification
Output : Updated solution *S′*, a flag *feasible* to indicate whether the modified solution is feasible
Step 0: Initialization. Put $FL \leftarrow NL$;
Step 1: Check if there are any remaining affected nodes
If (*FL* is empty)
Return *feasible* = <*true*> and updated solution *S′*;
Stop;
Else
Set node *i* = first node in *FL*;
Put $FL \leftarrow FL \backslash \{i\}$; //Remove the first node from *FL*;
Step 2: Update node information based on route path
Identify the node *j* that is visited by the same vehicle *k* right before node *i*;
Set $t_i' = t_j + \sigma_j + T_{ij}$;
If ($t_i' < a_i$), then $t_i' = a_i$;
Else if ($t_i' > b_i$) then return *feasible* = <*false*>; stop;
If (node *j* is a pickup node for customer *c*)
$q_i' = q_j + L_c$;
If ($q_i' > Q_k$) then return *feasible* = <*false*>; stop;
Else if (node *j* is a delivery node for customer *c*)
$q_i' = q_j - L_c$;
Step 3: Update node service start time based on shipment path
If (node *i* is a transshipment pickup node for customer *c*, that is, $i \equiv i_{ct}$), then
Find node *j* that is the corresponding transshipment delivery node for customer *c*, that is, $j \equiv j_{ct}$;
If ($t_j + \sigma_j > t_i'$) then set $t_i' = t_j + \sigma_j$;
If ($t_i' > b_i$) then return *feasible* = <*false*>; stop;
Step 4: Propagation
If ($t_i' = t_i$ and $q_i' = q_i$)
Set $t_i = t_i'$, $q_i = q_i'$;
Identify that node *j* that is visited by the same vehicle *k* right after node *i*;
Put $FL \leftarrow FL \cup \{j\}$ \\Add j to *FL*
If (node *i* is a transshipment delivery node for customer *c* and transshipment location *t*, , that is, $i \equiv j_{ct}$)
Find node *j* that is the corresponding transshipment pickup node for customer *c* and transshipment location *t*, that is, $j \equiv i_{ct}$;
Put $FL \leftarrow FL \cup \{j\}$;
Go to step 1;

Figure 2.10. Pseudocode for updating and checking feasibility for the routes

Phase I

In Phase I, a predefined number n_I^{max} of feasible solutions to the PDPT are iteratively constructed using randomness and a greedy objective function to guide the choices at each step. We start with a set of null routes, one for each vehicle $k \in K$, and an open pool U of all unsatisfied shipment requests. We then insert requests sequentially into the routes, but do not limit construction to one route at a time. Rather the routes are formed in parallel. At each step, an unassigned request is added to the partial solution by selecting at random from the ones that would increase the cost of the new solution the least, as measured by a greedy function. This is the essence of GRASP.

The `Basic_Insertion` procedure for a single request $i \rightarrow j$ is outlined in Figure 2.11. Given the current partial solution denoted by S_c, it is called for each element in the pool U, and for each such element, stores the n^{max} best insertions found returning them via the set SL. The elements of SL are used to construct a candidate list, $CL(S_c)$, which is used to select the next request to insert and to determine its pickup and delivery positions in the routes. To measure the "cost" of selecting one of the elements in $CL(S_c)$ at the next iteration, we define a greedy objective function $COST(S)$ for each partial solution S. Based on extensive testing, this function was selected to be the weighted sum of the travel distance D_{ij} for request $i \rightarrow j$, the indictor variable I_r which is equal to 1 if route r is not empty and 0 otherwise, and the indicator variable U_c which is equal to 1 if customer c's request is not satisfied by the routes in S and 0 otherwise; that is,

$$COST(S) = w_1 \sum_{r \in R} \sum_{(i,j) \in r} D_{ij} + w_2 \sum_{r \in R} I_r + w_3 \sum_{c \in C} U_c \qquad (2.4)$$

where w_1, w_2 and w_3 are the weights for total travel distance, number of nonempty routes and number of unsatisfied customer requests in S, respectively. In our implementation, $w_1 << w_2 << w_3$, so hierarchically speaking, the first objective of our greedy function is to cover all requests, the second is to use as few vehicles as possible, and the third is to minimize travel distance.

Procedure `Basic_Insertion`
Input : Shipment request $i \rightarrow j$, current partial solution S_c, maximum number of solutions n^{max} to return
Output : Set of solutions *SL* created by feasible insertions of $i \rightarrow j$ into routes of S_c
Step 0: Initialization: Put $SL \leftarrow \varnothing$;
Step 1: Single route insertions
For (all $r \in R$)
For (all insertion positions for $i \rightarrow j$ in route r)
Set $S_0 = S_c$, perform single route insertion on S_0;
Put $NL \leftarrow \varnothing$, add node i and j to *NL*;
If (`Check_Feasibility`(S_0, *NL*)), then
Remove $i \rightarrow j$ from open shipment pool of S_0;
If ($|SL| < n_I^{max}$), then add S_0 to *SL*;
Else //find worst solution in *SL* and replace with S_0
$S_{worst} = \text{argmax}_{S \in SL}(COST(S))$;
If ($COST(S_{worst}) > COST(S_0)$), then
Remove S_{worst} from *SL*, add S_0 to *SL*;
Step 2: Double route insertions
For (all transshipment pickup and delivery nodes t, t' for the customer associated with $i \rightarrow j$)
For (all $r_1 \in R$)
For (all $r_2 \in R$)
For (all insertion positions for $i \rightarrow t'$ in route r_1 and $t \rightarrow j$ in route r_2)
Set $S_0 = S_c$, perform double route insertion on S_0;
Put $NL \leftarrow \varnothing$, add nodes i, t, t' and j to *NL*;
If (`Check_Feasibility`(S_0, *NL*)), then
Remove $i \rightarrow j$ from open shipment pool of S_0;
If ($|SL| < n^{max}$), then add S_0 to *SL*;
Else //find worst solution in *SL* and replace with S_0
$S_{worst} = \max_{S \in SL}(COST(S))$;
If ($COST(S_{worst}) > COST(S_0)$), then
Remove S_{worst} from *SL*, add S_0 to *SL*;
Return *SL*;
Stop;

Figure 2.11. Pseudocode for basic insertion

In Steps 1 and 2 single and double insertions are performed and the best n^{max} solutions are returned in the set *SL*. In Step 1, all possible insertion positions for each of the $|K|$ routes are evaluated giving a time complexity of $O(|K| \cdot |N|^2)$. Recalling that the `Check_Feasibility`

procedure runs in $O(|N|)$ time, the complexity of a single insertion operation is $O(|K| \cdot |N|^3)$. In Step 2, all possible insertion positions in all pairs of routes for each transshipment location are evaluated. This can be done in $O(|T| \cdot |K|^2 \cdot |N|^4)$ time so with the feasibility check the complexity of the double insertion operation becomes $O(|T| \cdot |K|^2 \cdot |N|^5)$. Collectively, then, the overall complexity of the `Basic_Insertion` procedure is $O(|K| \cdot |N|^3 + |T| \cdot |K|^2 \cdot |N|^5) = O(|T| \cdot |K|^2 \cdot |N|^5)$.

At each step during construction, $CL(S_c)$ is sorted in nondecreasing order of $COST(S)$ and one element is picked at random from the top l candidates to obtain an augmented partial solution. The process is repeated until all customer requests are satisfied or no more candidates can be found. The pseudocode for route construction is given in Figure 2.12.

Procedure `Route_Construction`

Input : Customer requests U, available vehicles K

Output : Solution S_c with all customer requests routed

Step 0: Initialization. $S_c \leftarrow \varnothing$; //All routes empty, all requests in U

Step 1: Build candidate list CL

Put $CL \leftarrow \varnothing$;

For ($i \rightarrow j \in U$)

Call `Basic_Insertion`($i \rightarrow j$, S_c) and set solution list $SL = S_c$;

//Add solutions in SL to CL

For ($S_0 \in SL$)

If ($|CL| < l$), then put $CL \leftarrow CL \cup S_0$;

Else //find worst solution in CL and replace with S_0

$S_{worst} = \max_{S \in CL}(COST(S))$;

If ($COST(S_{worst}) > COST(S_0)$), then

Remove S_{worst} from CL, add S_0 to CL;

Step 2: If ($CL = \varnothing$), then

Return current solution S_c;

Stop;

Step 3: Select a new solution S randomly from CL with probability of $1/|CL|$;

Remove request $i \rightarrow j$ associated with S from U;

Set $S_c = S$;

Go to Step 1.

Figure 2.12. Pseudocode for route construction

In `Route_Construction`, shipment requests are inserted into routes sequentially. To determine the next insertion, all requests in the open shipment pool U are evaluated with the `Basic_Insertion` procedure, which runs in $O(|T| \cdot |K|^2 \cdot |N|^5)$ time. Because the candidate list has to be build for each partial solution S_c which means $O(|C|)$ times, and there are $|C|$ requests, we can conclude that the overall complexity of Phase I is $O(|T| \cdot |K|^2 \cdot |N|^5 \cdot |C|^2)$.

Although polynomial, this value is high and may foreshadow poor performance. To avoid redundant computations and hence reduce runtimes, we have implemented a hashing procedure which is discussed later in "performance improvement" section.

Phase II

In Phase II, an attempt is made to improve a subset of the Phase I solutions using ALNS. The general idea is to remove some requests from routes in the current solution and then reinsert them elsewhere to arrive at a new solution. At each iteration, one each of several removal and insertion heuristics is randomly selected based on their designated weights. Letting the set of removal heuristics be $\{rh_1, \ldots, rh_{nr}\}$ and their corresponding weights be $\{rw_1, \ldots, rw_{nr}\}$, rh_i is chosen with probability $rw_i / \sum_{j=1}^{nr} rw_j$, where nr is the number of options. Similarly for the insertion heuristics $\{ih_1, \ldots, ih_{ni}\}$ and their corresponding weights $\{iw_1, \ldots, iw_{ni}\}$, the probability of ih_i being selected is $ih_i / \sum_{j=1}^{ni} ih_j$, where ni is the number of options. After each iteration, if the solution realized is feasible and better than the incumbent, the latter is updated. In any case, the process is repeated until a predefined number of iterations n_{II}^{max} or a time out limit T_{II}^{max} is reached.

The basic steps of the ALNS are given Figure 2.13. In the following sections, we discuss each removal and insertion heuristic in detail. This includes the procedures for randomly selecting the requests and adaptively updating the heuristic weights.

Procedure ALNS

Input: Current solution S_c, maximum number of iterations n_{II}^{max}, time out period T_{II}^{max}, number of requests to be removed μ, removal heuristics $\{rh_1, \ldots, rh_{nr}\}$, insertion heuristics $\{ih_1, \ldots, ih_{ni}\}$

Output: Best solution found S_b

Step 0: Initialization

Set *iter* = 1, $S_i = S_c$, $S_b = S_c$, run time $T_{run} = 0$;

Set number of requests to be removed at each iteration to μ;

Set initial removal heuristics weights $\{rw_1, \ldots, rw_{nr}\}$;

Set initial insertion heuristics weights $\{iw_1, \ldots, iw_{ni}\}$;

Step 1: Set $S = S_i$;

Step 2: Randomly select a removal heuristic rh_i based on their corresponding weights and remove μ requests from routes of S; add them to the open shipment pool U;

Step 3: Randomly select an insertion heuristic ih_i based on their corresponding weights and insert open requests in U to routes of S; //infeasible solutions have very high costs

Step 4: If ($COST(S) < COST(S_i)$), then set $S_i = S$;

If ($COST(S) < COST(S_b)$), then set $S_b = S$;

Update $\{rw_1, \ldots, rw_{nr}\}$, $\{iw_1, \ldots, iw_{ni}\}$, μ;

Put *iter* $\leftarrow$ *iter* + 1;

Update T_{run};

If (*iter* > n_{II}^{max} or $T_{run} > T_{II}^{max}$), then return S_b;

Go to Step 1.

Figure 2.13. Pseudocode for ALNS

Removal heuristics

Our version of ALNS includes three removal heuristics. In each case, μ requests are removed from the current set of routes and placed in the open pool U. To increase the versatility of the approach, the parameter μ is adaptively updated based on how well a given value performed on previous iterations. If a shipment requests $i \rightarrow j$ is selected for removal, it could be reinserted into a single route, or into multiple routes using transshipment locations. In either case, nodes i, j and any transshipment nodes between the two are removed allowing us to reconstruct the path using

other transshipment locations or simply eliminating them for this request. For example, assume that the shipment path for customer c_1 is $\{ i_{c_1}, j_{c_1} \}$ and that the shipment path for customer c_2 is $\{ i_{c_2}, j_{c_2 t}, i_{c_2 t}, j_{c_2} \}$, implying that location t is a transshipment point for customer c_2's load. The requests that can be removed from routes are $i_{c_1} \to j_{c_1}$, $i_{c_2} \to j_{c_2 t}$, $i_{c_2 t} \to j_{c_2}$ and $i_{c_2} \to j_{c_2}$. If $i_{c_1} \to j_{c_1}$, then the nodes i_{c_1} and j_{c_1} from customer c_1's route and $i_{c_1} \to j_{c_1}$ is added to the open pool U. The same process is followed if either $i_{c_2} \to j_{c_2 t}$ or $i_{c_2 t} \to j_{c_2}$ is removed. If $i_{c_2} \to j_{c_2}$ is removed, then all four nodes i_{c_2}, $j_{c_2 t}$, $i_{c_2 t}$ and j_{c_2} are removed from customer c_2's routes and $i_{c_2} \to j_{c_2}$ is added to U.

It is worth noting that we allow customer requests to be handled in segments when their routes include transshipment points. As the example indicates, if $i_{c_2 t} \to j_{c_2}$ is removed, the partial route $\{ i_{c_2}, j_{c_2 t} \}$ still remains for customer c_2. The request $i_{c_2 t} \to j_{c_2}$ is placed in the open pool to be reinserted into a route at a later time. Using the weighting described above, one of the following the three removal heuristics is selected randomly at each iteration.

Shaw removal heuristic (rh_1) tries to select requests that are in some sense similar to each other. In so doing, there is a higher likelihood that they can be reinserted into different routes giving an overall cost reduction. If the selected requests are significantly different from each other, it's very likely that to retain feasibility, most if not all of them will have to be reinserted into their original positions, yielding no improvement.

The heuristic consists of three operations: (1) a request is randomly removed from the existing routes and placed in the set *RS*, (2) the requests not in *RS* are sorted using a relatedness measure $R_{i \to j}$ that is defined presently, and (3) one request is randomly selected from the sorted list using probabilities derived from $R_{i \to j}$. This process is repeated until the desired number of requests μ are removed.

Shaw (1998) proposed the following relatedness measure for request $i \to j$ associated with customer c_1 currently on a route,

$$R_{i \to j} = \sum_{k \to l \in RS} \left(\theta_1 \left| L_{c_1} - L_{c_2} \right| + \theta_2 \left| D_{ik} + D_{jl} \right| + \theta_3 \left(\left| t_i - t_k \right| + \left| t_j - t_l \right| \right) \right) \tag{2.5}$$

where c_2 is the customer associated with shipment request $k \rightarrow l$, L_{c_1} and L_{c_2} are the loads for customer c_1 and c_2, respectively, D_{ik} is the distance from node i to k, t_i, t_j, t_k, and t_l are the respective route service start times for nodes i, j, k, and l in the current solution, and θ_1, θ_2 and θ_3 are weights for each component in Eq. (2.5). By definition, the smaller the value of $R_{i \rightarrow j}$, the more similar request $i \rightarrow j$ is to requests in *RS*.

To implement step 3, an array *AR* is defined that holds the requests that have not been removed from the existing routes. The elements in *AR* are sorted such that if $i < j$, than $R_{AR[i]} < R_{AR[j]}$. We then randomly generate a number $p \in (0, 1)$, and for a predefined parameter $\alpha \geq 1$, we choose element $AR[p^{\alpha} \cdot |AR|]$ to be added to *RS*. If $\alpha = \infty$, then the request with smallest relatedness measure is always selected; if $\alpha = 1$, the request is picked randomly.

Random removal heuristic (rh_2) randomly selects shipment requests in routes to remove. Randomness introduces a degree of diversity that helps surmount local optima.

Route random sweep heuristic (rh_3) randomly picks a route and removes all shipment requests on the route. This process is repeated until any additional removals result in more requests being removed than specified by the current value of μ. In that case, the *Shaw removal heuristic* is used to bring the number up to μ. By removing a complete route at each step, we increase the possibility of finding a new solution that requires fewer vehicles than the incumbent.

Insertion heuristics

The requests in the open pool *U* are first ordered and then inserted into routes using one of four heuristics. The procedure is given in Figure 2.8 where we saw that the insertion position was determined by a greedy objective function *COST*(*S*) given in Eq. (2.4). In order to describe each heuristic, we need a bit more notation. For request $i \rightarrow j$, let $IN_{(i,j)}$ contain all feasible insertions and let Λ_{ij} be an element of $IN_{(i,j)}$. For the single insertion case, each $\Lambda_{ij} \in IN_{(i,j)}$ is determined by the target route *r* and the corresponding positions of *i* and *j* in the sequence; for the double insertion case, it is determined by the target routes *r* and *r′*, the positions of *i* and *j*, and the transshipment nodes *t* and *t′* used in *r* and *r′*, respectively.

Greedy insertion heuristic (ih_1) evaluates all shipment requests in *U* with respect to all possible insertion positions in all routes in a solution *S* one at a time. Letting the current objective value be $C \equiv COST(S)$, the new objective value after the single insertion Λ_{ij} is defined

as $C_{\Lambda_{ij}}$. The best insertion, call it Λ^*, and corresponding shipment request $i^* \rightarrow j^*$ is determined by $(\Lambda^*, i^* \rightarrow j^*) = \arg\min_{i \rightarrow j \in U, \Lambda_{ij} \in IN_{(i,j)}} \left\{ C_{\Lambda_{ij}} - C \right\}$. After Λ^* is performed, the routes in the corresponding solution are updated and the request $i^* \rightarrow j^*$ is removed from *U*. This process is repeated until the pool is empty or no request in the pool can be inserted into a route.

Regret-k insertion heuristic (ih_2) selects the request that would produce the greatest regret if it were not inserted first. For a given $i \rightarrow j$, all possible insertion positions are evaluated and the *k* best positions are identified again using the greedy objective function. Letting $\Lambda_{ij}^1, \ldots, \Lambda_{ij}^k$ denote these positions, we then sort them such that $C_{\Lambda_{ij}^m} \leq C_{\Lambda_{ij}^n}$ if $m < n$. For all shipment requests in the pool *U*, we pick $i^* \rightarrow j^* = \arg\max_{i \rightarrow j \in U} \left\{ C_{\Lambda_{ij}^k} - C_{\Lambda_{ij}^1} \right\}$. The best insertion $\Lambda_{i^* j^*}^1$ is performed, the solution is updated, and the process is repeated until $U = \varnothing$.

Random insertion (ih_3) randomly decides the order in which requests are inserted.

Most constrained first insertion (ih_4) determines the insertion order of the requests in *U* based on a weighted function of the distance between supply and demand locations, time windows and shipment load. The corresponding function is

$$Q_{i \rightarrow j} = \beta_1 D_{ij} + \beta_2 / \left(\left| b_i - a_i \right| + \left| b_j - a_j \right| \right) + \beta_3 L_c \qquad (2.6)$$

where β_1, β_2 and β_3 are weights for each component in (2.6) and *c* is the customer associated with request $i \rightarrow j$. The larger the value of $Q_{i \rightarrow j}$, the more likely $i \rightarrow j$ is to be selected for insertion. This logic is motivated by the fact that if a request is hard to accommodate, we want to consider it first. That is, requests that require longer travel distances, have shorter service time windows, or have larger shipment loads are more difficult to fit into a route.

Similar to the Shaw removal heuristic, to select the next request to insert, we place the candidates in the open pool *U* in a sorted array *AR* such that if $i < j$, than $Q_{AR[i]} > Q_{AR[j]}$. We then randomly generate a number $p \in (0, 1)$ and for a predefined parameter $\alpha \geq 1$, we choose element $AR[p^{\alpha} \cdot |U|]$ to be inserted. If $\alpha = \infty$, then the most constrained request is always picked first; if $\alpha = 1$, the request is picked randomly. The process is repeated until $U = \varnothing$ or the remaining requests cannot be feasibility inserted into any of the $|K|$ routes.

Regardless of the heuristic, a request can be inserted into either an existing route or multiple routes with a transshipment node on the path from *i* to *j*. As mentioned, introducing the transshipment option offers the possibility of temporarily freeing up capacity on a vehicle to be used in the interim to transport shipments to and from nearby locations, or the possibility of connecting shipments on different routes. The disadvantage, though, is that the size of the network and hence the number of routes greatly increase making the computations much more difficult.

A second issue relates to the sequential nature of the insertion heuristics. For each request $i \rightarrow j$, we use `Basic_Insertion` to find the best positions in the current set of routes to insert it. All possible single route insertion and double route insertion positions are considered. In most cases, the best route is realized from the single route insertion because visiting a transshipment location usually results in longer travel distances. If we were to consider two or more requests simultaneously, the use of transshipment locations might produce shorter total distances than if each request were considered sequentially. To overcome this shortcoming, we disable single route insertion with a predefined probability $p \in (0, 1)$ when evaluating routes that have yet to include a transshipment location.

Adaptive parameter adjustment

As suggested by Ropke and Pisinger (2006), we also adaptively change the insertion and removal weights, and hence probabilities, of the corresponding heuristics at each step based on the relative performance of each as the `ALNS` procedure progresses from one segment to the next, where a *segment* is a predefined number of iterations indexed by *s*. Within a segment, we keep track of the number of times that heuristic *h* is used (call it γ_h) and its "score" (call it π_h), which is a qualitative measure. At the beginning of each new segment, γ_h and π_h are reset to 0 for all *h*. At each iteration, when a particular removal heuristic rh_i and a particular insertion heuristic ih_j are used, their counters, γ_{rh_i} and γ_{ih_j}, are incremented by 1. Also, if the solution obtained at that iteration is better than the incumbent, the scores π_{rh_i} and π_{ih_j} are incremented by δ_1; if the solution is new but inferior to the incumbent, we increment them by δ_2. In all other cases, the scores are left unchanged. At the end of a segment, we update all weights using the reaction factor $\rho \in (0, 1)$ which determines the rate of change according to the performance of the

heuristic. For segment *s*, the new removal heuristic weights are $rw_i^s = (1-\rho)rw_i^{s-1} + \rho\pi_{rh_i} / \gamma_{rh_i}$, i = 1, 2,…, *nr*, and the new insertion heuristic weights are $iw_j^s = (1-\rho)iw_j^{s-1} + \rho\pi_{ih_j} / \gamma_{ih_j}$, j = 1, 2,…,*ni*.

One of the ways our method differs from previous approaches is that we reactively change the number of customer requests to be removed at each iteration. A similar scheme was introduced by Battiti and Tecchioli (1994) whose reactive mechanism was based on the tabu tenure feature of tabu search that determines how long a move is forbidden before it is allowed to reappear. Reactive tabu search uses a dynamic systems theory analogy, where the tabu tenure depends on the repetition of solutions rather than fixed in length. In our approach, the number of requests to be removed, μ, is selected from a set of *m* numbers arrayed in ascending order: $\{\mu_1, \mu_2,...,\mu_a,...,\mu_m\}$. `ALNS` starts with the average number $\mu = \mu_a$ and once a better solution is found, μ is reset to μ_1 to intensify the local search; now only a small number of requests are removed at each iteration. As a diversification mechanism, if the number of times all previous solutions are realized is λ_1 in a predefined number of iterations (say, λ_2), then μ is set to the next highest value, and so on until $\mu = \mu_m$ is reached. If the solution uncovered at the current iteration is better than the incumbent, μ is reset to μ_a so long as $\mu > \mu_a$.

The general idea is that if a good solution is found, we would like to intensify the search around it to try to find a local minimum. This is best achieved with a small value of μ. Once it is determined that the algorithm is trapped in a neighborhood around a local minimum, the search is expanded by increasing μ. The larger the value of μ, the more diversified the neighborhood being searched. However, larger neighborhoods mean longer search times per iteration, so one needs to be careful in selecting μ_m.

Performance improvement

When using insertion heuristics to construct and improve routes, it is often necessary to evaluate the same potential move repeatedly. In our case, to reduce the amount of effort wasted on duplicate calculations, it is beneficial to save the best insertion positions for a given request and route rather than recompute them at each iteration. This was achieved by implementing cache structures using a hash table. One cache was created for the best insertion positions for single route insertions and a second for best insertion positions for double route insertions. In the design,

a hash key is generated for each request and the routes in which it can be inserted. During the course of the algorithm, when an insertion needs to be evaluated, a check is first made to see if its hash key exists, and if so, the corresponding best insertion position is used. Otherwise, all possible routes and insertion positions are evaluated and the best are saved in cache.

For a single insertion, we have $key_s = \{r = \{b, i_1, \ldots, i_n, b'\}, st = \{t_b, t_{i_1}, ..., t_{i_n}, t_{b'}\}, i \rightarrow j\}$ which consists of three components: the route r as represented by a sequence of nodes, service start times st for the nodes in the route, and the request $i \rightarrow j$. Two keys are the same only when their corresponding components r, st and $i \rightarrow j$ are the same. Similarly, for a double insertion, we have $key_d = \{r_1 = \{b, i_1, \ldots, i_n, b'\}, r_2 = \{b, j_1, \ldots, j_m, b'\}, st_1 = \{t_b, t_{i_1}, ..., t_{i_n}, t_{b'}\}, st_2 = \{t_b, t_{j_1}, ..., t_{j_m}, t_{b'}\}, i \rightarrow j\}$ which has five components: the first route r_1 and its service start times st_1, the second route r_2 and its service start times st_2, and the shipment request $i \rightarrow j$. Two keys are the same only when their corresponding components r_1, r_2, st_1, st_2 and $i \rightarrow j$ are the same. The service start times in the keys are necessary because of transshipment. The following example demonstrates the benefit of using cache.

Example 2.3. Let the depot start and end nodes be labeled 0 and 1, and assume there are 3 customers with corresponding pickup and delivery nodes 2, 3, 4, 5, 6 and 7. Also assume that two vehicles are available and that transshipments are not permitted. For simplicity, we will also ignore the time window and vehicle capacity constraints. Table 2.2 provides the data necessary to assess the benefit of using cache in the construction phase for three iterations. The first column is the iteration number; the second column contains two entries−−the first is a route as represented by a sequence of nodes visited, and the second is the shipment request to be inserted into the route. The third column lists the routes that result from inserting each request, while the fourth column lists the entries added to the cache at the corresponding iteration. The hash key is the route-request pair and the hash value is the best route that results considering all possible insertions. The last column identifies the selected route from all the candidates.

The construction phase starts with an initial solution containing only empty routes {0, 1} and {0, 1} and the open shipment pool of the solution contains three shipment requests; that is, $U = \{2 \rightarrow 3, 4 \rightarrow 5, 6 \rightarrow 7\}$. At iteration 1, all requests are considered for insertion into the empty routes. Because the two routes are identical, only one needs to be evaluated. For request $2 \rightarrow 3$,

the route created is {0, 2, 3, 1}. Because this request-route pair has not yet been evaluated, we update the cache to include this candidate. The cache key is composed of request 2→3, route {0, 1} and the service start time for the nodes in route. For simplicity, the service start times are not shown. The cache value is the best route found, in this case {0, 2, 3, 1}. The other two request-route pairs are processed in a similar fashion giving routes {0, 4, 5, 1} and {0, 6, 7, 1}. The results are also stored in the cache.

Assuming that the route {0, 4, 5, 1} has the lowest cost among the three candidates, the solution is updated and now contains the two routes {0, 4, 5, 1} and {0, 1}. The open shipment pool has two requests 2→3 and 6→7 remaining. At iteration 2, the insertions of these requests into route {0, 1} have already been evaluated so we can retrieve the best routes from the cache without additional calculations. For shipment request 2→3 and route {0, 4, 5, 1}, we have 6 insertion positions to evaluate as shown in the third column of Table 2.2. Assume that the best route is {0, 4, 5, 2, 3, 1}. Because this request-route pair has not yet been evaluated, we save the result in the cache. The hash key in this case consists of request 2→3, the route {0, 4, 5, 1} and its service start times. The hash value is the route {0, 4, 5, 2, 3, 1}. Similarly for request 6→7 and route {0, 4, 5, 1}, there are again 6 insertion positions to evaluate. The best route {0, 6, 4, 7, 5, 1} is saved in the cache. Assuming that the route {0, 2, 3, 1} is selected from the three best candidates, the routes in the solution after this iteration are {0, 4, 5, 1} and {0, 2, 3, 1} as indicated in the last column.

At this point, there is only one request 6→7 left unsatisfied. An iteration 3, the insertion of 6→7 into route {0, 4, 5, 1} has been evaluated in iteration 2, so we can retrieve the best route for this pair from the cache. What remains is to evaluate the six insertion positions for request 6→7 and route {0, 2, 3, 1}. Assuming that the best route that results from these six possibilities is {0, 2, 3, 6, 7, 1} and that it has higher cost than {0, 6, 4, 7, 5, 1}, we update the cache in column 4 accordingly, and include the route in column 5. At the conclusion of this iteration, all requests are satisfied so the final solution is given by {0, 6, 4, 7, 5, 1} and {0, 2, 3, 1}.

As shown in the second column of Table 2.2, without using cache, we would have to evaluate all nine route-request combinations. This corresponds to the 29 tentative routes shown in the third column. The bold entries in the second column identify the route-request pairs that were previously evaluated, so with cache, the best routes for each can be retrieved without

duplicate calculations. Thus, only 6 route-request combinations and 21 insertion positions need to be evaluated. Testing showed that the use of cache reduced Phase I and Phase II runtimes by at least 95%. ■

Table 2.2. Example of benefit using cache in construction phase

Iter.	Evaluated route and request pair	Evaluated routes (insertion positions)	Best route stored in cache (key -- value)	Selected route (best)
1	{0, 1}, 2→3	{0, 2, 3, 1}	{0, 1}, 2→3 -- {0, 2, 3, 1}	{0, 4, 5, 1}
	{0, 1}, 4→5	{0, 4, 5, 1}	{0, 1}, 4→5 -- {0, 4, 5, 1}	
	{0, 1}, 6→7	{0, 6, 7, 1}	{0, 1}, 6→7 -- {0, 6, 7, 1}	
2	**{0, 1}, 2→3**	{**0, 2, 3, 1**}		{0, 2, 3, 1}
	{0, 1}, 6→7	**{0, 6, 7, 1}**		
	{0, 4, 5, 1}, 2→3	{0, 2, 3, 4, 5, 1} {0, 2, 4, 3, 5, 1} {0, 2, 4, 5, 3, 1} {0, 4, 5, 2, 3, 1} {0, 4, 2, 3, 5, 1} {0, 4, 2, 5, 3, 1}	{0, 4, 5, 1}, 2→3 -- {0, 4, 5, 2, 3, 1}	
	{0, 4, 5, 1}, 6→7	{0, 6, 7, 4, 5, 1} {0, 6, 4, 7, 5, 1} {0, 6, 4, 5, 7, 1} {0, 4, 5, 6, 7, 1} {0, 4, 6, 7, 5, 1} {0, 4, 6, 5, 7, 1}	{0, 4, 5, 1}, 6→7 -- {0, 6, 4, 7, 5, 1}	
3	**{0, 4, 5, 1}, 6→7**	{**0, 6, 7, 4, 5, 1**} {**0, 6, 4, 7, 5, 1**} {**0, 6, 4, 5, 7, 1**} {**0, 4, 5, 6, 7, 1**} {**0, 4, 6, 7, 5, 1**} {**0, 4, 6, 5, 7, 1**}		{0, 6, 4, 7, 5, 1}
	{0, 2, 3, 1}, 6→7	{0, 2, 3, 6, 7, 1} {0, 2, 6, 3, 7, 1} {0, 2, 6, 7, 3, 1} {0, 6, 7, 2, 3, 1} {0, 6, 2, 3, 7, 1} {0, 6, 2, 7, 3, 1}	{0, 2, 3, 1}, 6→7 -- {0, 2, 3, 6, 7, 1}	

Data Set Generation

One challenge we faced in evaluating the performance of the proposed GRASP was the lack of publically available data sets to benchmark against. As an alternative, we developed our own PDPT instances with 25 requests and 1 transshipment location each. We start by randomly generating PDP instances with 6 requests each and add a transshipment location. Although these instances turned out to be too large for CPLEX to handle, they were solvable with our branch and price code (unpublished). We then increased the number of requests in each instance to 25 using the methodology described below which ensures that the optimal objective function values of the paired instances are the same. This provided a direct way to evaluate the solution quality of the GRASP.

Specifically, given an optimal set of routes for a PDPT, if an additional pickup or delivery location is inserted along the trajectory of one of the routes, under certain conditions, the modified route will be the same length as the original. Since we want the transshipment

location to play a role in the solution, we need to consider the trajectories of two routes at a time. In particular, there are three cases in which the trajectories of two routes may intersect:

Case 1: A customer pickup or delivery location in one route might be the same as a pickup or delivery location of a different customer in another route

Case 2: Two routes might use the same transshipment location *t*

Case 3: The two routes overlap but do not share a common pickup or delivery location.

Figure 2.14 illustrates the first two cases; Figure 2.15 illustrates the third. We now explain how we can add a transshipment location *t* at the intersection of the two routes if one does not already exist. Assuming that the vehicle traversing route r_1 reaches the intersection before the vehicle traveling along route r_2 reaches it, we can randomly select a point along r_1's trajectory between the depot and the intersection and designate it as a pickup location for a new customer *c*. Similarly, we can randomly select a location along route r_2's trajectory after the intersection and make it a delivery location for the same customer. The time windows for the new pickup and delivery nodes, call them *i* and *j*, are based on the arrival times, given the current solution. To limit the changes of creating an infeasible instance, a random buffer is added around the arrival times that is a function of the input parameter τ^{win}. Letting node *i′* be the immediate predecessor of node *i* on the modified route, we calculate that the vehicle will arrive at node *i* at $t_i = t_{i'} + \sigma_{i'} + T_{i'i}$. As such, we set $a_i = t_i - p \cdot \tau^{win}/2$ and $b_i = t_i + p \cdot \tau^{win}/2$, where *p* is a random number between 0 and 1. The demand L_c is also randomly generated. Assuming customer *c* is on the route served by vehicle *k*, we set $L_c = \lceil p \cdot Q_k \rceil$.

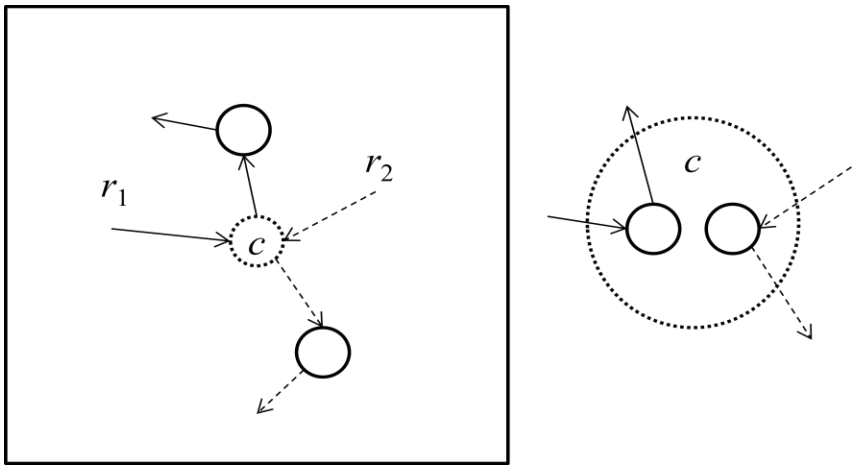

Figure 2.14. Cases 1 and 2: two routes intersecting at the same location *c*.

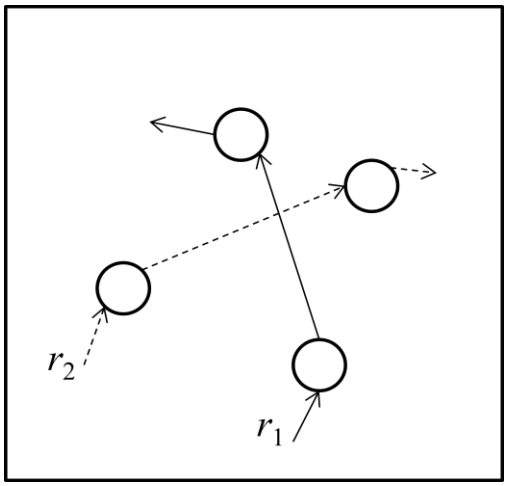

Figure 2.15. Case 3: trajectories of two overlapping routes that do not share a common location

The augmented routes are then checked see if they are feasible with respect to the time windows and vehicle capacity. If they are, the new customer is added to the data set and the solution is updated to include the new pickup and delivery nodes in the routes; otherwise, the customer is discarded. The process is repeated until the desired number of new customer requests is created. Of course, the generation procedure can be used to create one or more PDPT instances with known lower bounds from an existing PDP instance by modifying the routes of its best know solution.

Proposition 2.4. Given a PDP instance and its optimal solution *S*, the data set construction method creates a new instance with an optimal objective function value no greater than *COST*(*S*).

Proof. The cost function (1) consists of three terms: total distance traveled, number of nonempty routes, and number of unsatisfied customer requests in *S*. The data set construction method works by adding new requests along the trajectories of routes in *S*. In so doing, it constructs a solution $\bar{S}$ which contains routes that have same trajectories as those in *S* while satisfying all capacity and time window constraints. By design, the total distance traveled is the same as in *S* and all new customer requests are served in $\bar{S}$. Therefore, the number of open customer requests is the same as in *S* as is the number of vehicles required to serve the new routes. By implication, *COST*($\bar{S}$) = *COST*(*S*), and since $\bar{S}$ is feasible, the optimal objective function of the new instance is no worse than COST($\bar{S}$), therefore no worse than COST(*S*). ■

2.6 COMPUTATION RESULTS

2.6.1 Branch and price

The branch and price algorithm was implemented in C++, compiled with visual studio 2008, and run on a notebook with 2 GB of memory and an Intel Core Duo CPU with a pair of 2 GHz cores. The restricted master problems are solved using CPLEX 10. The test instances are generated randomly. The transshipment locations are generated at the central locations of the request clusters. The test instances created are named as T_# of requests_# of transshipment locations_test case number. We first exam the strength of the bound obtained using column generation in comparison to the one obtained by solving LP relaxation of the original arc-flow formulation model (2.1). In Table 2.3, Column 1 is the names of the test data instances. Column 2 contains the optimal solution objective Z_{IP} for the original arc-flow formulation solved using CPLEX. In column 3 we show Z_{LP}^1 the solution objective of LP relaxation of the original formulation and the corresponding LP Gap, where the Gap is calculated as $100\times(Z_{IP} - Z_{LP})/ Z_{IP}$. Similarly, in column 4, Z_{LP}^2 the solution objective by solving LP relaxation of the master problem using column generation is shown, along with the corresponding Gap and number of columns needed to solve the relaxed master problem.

Table 2.3. Strength of column generation

Test case	Optimal solution Z_{IP}	LP relaxation		Restricted master problem		
		Z_{LP}^1	Gap (%)	Z_{LP}^2	Gap (%)	No. of columns
T_2_1_1	2305.55	1083.25	53.20	2305.55	0.00	8
T_2_1_2	1166.22	1166.22	0.00	1166.22	0.00	7
T_2_1_3	2322.58	1590.69	31.51	2322.58	0.00	5
T_2_1_4	1957.95	1379.65	29.54	1957.95	0.00	4
T_2_1_5	1943.27	1415.53	27.16	1943.27	0.00	5
T_3_1_1	2211.65	794.37	64.08	1770.54	19.94	9
T_3_1_2	1934.25	1125.39	41.82	1704.43	11.88	17
T_3_1_3	2579.90	1122.28	56.50	2579.90	0.00	14
T_3_1_4	1145.42	725.77	36.64	1145.42	0.00	12
T_3_1_5	1644.28	905.51	44.93	1433.31	12.83	13
Average			38.54		4.47	

Based on our experience using a time limit of 2 hours, only the test cases with up to 3 customer requests can be solved to optimality using CPLEX directly. Consequently, our initial testing on the strength of LP relaxation based on the column generation was done on test cases with up to 3 customer requests. The comparison results show that the bounds obtained by column generation are considerably better than the ones from LP relaxation. The average LP Gap obtained by solving the model (2.1) directly is 38.54% while the Gap obtained by column generation is 4.47%.

In the following tests, we exam the performance of the branch and price algorithm on test cases with more than 3 customer requests. The test results are shown in Table 2.4. The names of test cases are given in column 1. The objective Z_{up} of initial primal solution obtained using the GRASP heuristic and the CPU time required t_{up} are shown in column 2 and 3 respectively. The objective of root node Z_{LP}^2 is given in column 4, along with the time spent for column generation of the root node t_{LP} in column 5. The final solution objective Z_{IP} and total CPU time required t_{IP} are given in column 6 and 7. Column 8 is the LP Gap between the LP relaxation of the root node and the optimal solution. Column 9 and 10 are the number of nodes evaluated in the branch and bound tree and the total number of columns generated respectively. The total solution time t_{tot} is

given in Column 11. The last column indicates that transshipment is used in the optimal solution with *.

Table 2.4. Branch and price results on test cases

Test case	Z_{up}	t_{up} (secs)	Z_{LP}^2	t_{lp} (secs)	Z_{IP}	t_{IP} (secs)	Gap	#nodes	#columns	t_{tot} (secs)	Trans-shipment
T_4_1_1	32754.96	1.2	22655.82	3.39	32754.96	1.2	30.83	81	2954	42.94	*
T_4_1_2	22552.99	1.12	22552.99	1.77	22552.99	1.12	0	1	65	1.77	
T_4_1_3	32885.11	1.02	27901.67	1.54	32885.11	1.02	15.15	3	116	2.36	
T_4_1_4	21878.7	0.82	21878.7	2.8	21878.7	0.82	0	1	525	2.81	
T_4_1_5	33345.92	4.83	33345.92	5.25	33345.92	4.83	0	1	123	5.25	
T_4_1_6	33515.92	0.76	33515.92	0.99	33515.92	0.76	0	1	19	0.99	
T_4_1_7	34087.02	1	34087.02	1.43	34087.02	1	0	1	42	1.43	*
T_4_1_8	33608.75	1.41	27423.36	6.05	33608.75	1.41	18.4	177	7520	179.99	*
T_5_1_1	21745.6	3.58	15684.6	55.03	21745.6	3.58	27.87	49	8516	349.38	
T_5_1_2	22210.45	5.24	19925.9	14.64	22210.45	5.24	10.29	3	418	22.99	
T_5_1_3	32402.48	1.34	32402.48	10.97	32402.48	10.97	0	1	946	10.98	
T_5_1_4	21945.04	1.83	20165.18	9.22	21945.04	77.91	8.11	17	4633	77.91	*
T_5_1_5	32874.96	1.68	22399.99	39.65	32874.96	41.77	31.86	455	43883	967.38	*
T_5_1_6	22048.7	3.15	16981.43	57.38	22048.7	60.39	22.98	63	23181	549.07	
T_5_1_7	32907.09	1.99	25270.89	3.7	32907.09	1.99	23.21	21	1818	15.43	
T_5_1_8	21981.75	2.43	17018.96	44.68	21981.75	2.43	22.58	3	1471	89.28	
T_5_1_9	22563.4	1.62	21044.43	19.99	22563.4	1.62	6.73	31	6861	229.43	
T_5_1_10	32787.41	1.29	32702.84	5.84	32787.41	1.29	0.26	7	1192	19.62	
T_6_1_1	21992.23	5.13	19234.18	769.68	21992.23	5.13	12.54	19	15763	2771.47	
T_6_1_2	43264.84	3.76	27538.24	23.82	43264.84	49.97	36.35	1237	73570	1636.45	
T_6_1_3	32695.93	2.59	32695.93	130.62	32695.93	2.59	0	1	1145	130.63	
T_6_1_4	32358.66	4.95	21548.96	41.15	22510.94	43.57	4.27	13	7601	247.72	*
T_6_1_5	22359.01	3.94	21576.68	62.2	22359.01	3.94	3.5	7	7296	246.8	*
T_6_1_6	32779.29	3.84	22626.18	42.4	32779.29	3.84	30.97	159	35365	961.62	
T_6_1_7	33176.82	4.2	25254.59	110.35	33144.96	115.71	23.81	781	132605	3059.31	*
T_6_1_8	43584.91	2.48	36085.38	36.81	43584.91	38.2	17.21	195	17201	590.85	*
T_6_1_9	22250.06	3.83	22250.06	461.08	22250.06	3.83	0	1	1402	461.09	
T_6_1_10	32682.82	4.07	24082.08	60.89	32682.82	4.07	26.32	867	140323	2389.32	

In general, branch and price can find optimal solutions for up to 6 customer requests within the 2-hour time limit. We had less success for larger instances. For the instances where we found the optimal solution in given time limit. We had the following observations: (1) In small test cases, we mostly obtained optimal solutions at the root node of the branch and bound tree. This conforms with our finding regarding the strength of the column generation. For larger instances, more nodes must be explored; (2) In about 1/3 of the test cases, we found that transshipment is beneficial; and (3) GRASP provides optimal solutions in most test cases in considerably less time.

2.6.1 GRASP

The GRASP was implemented in C++, compiled with visual studio 2008, and run on a notebook with 2 GB of memory and an Intel Core Duo CPU with a pair of 2 GHz cores. Testing was done for both PDP and PDPT data sets, recognizing that if the algorithm didn't perform well on the former, it would be difficult to justify its use on the latter. In the first set of experiments, we solved a series of benchmark PDP instances and compared our results with the best known solutions. In the second set of experiments, we used our random problem generator to create 50 instances with known optimal solutions and undertook similar comparisons. As mentioned in Section 2.2, Codato and Fischetti (2006), Shang and Cuff (1996), Thangiah et al. (2007), and Yan et al. (2006) all worked on variations of the PDPT but when contacted, they replied that their data were not available or they failed to respond after several inquires (we did get one of the instances investigated by Shang and Cuff and were able to significantly improve their solution).

For the two sets of experiments, results are reported for different fixed values of the removal-insertion parameter μ and for the approach where it is adjusted dynamically, as described in Section 2.5.2. This analysis allowed us to determine whether or not the reactive version of ALNS improves overall performance and yields better solutions. Results are also given for test runs under different Phase II iteration frequencies. In virtually all cases, the final solutions returned by Phase II significantly improved the best Phase I solutions.

Before running the GRASP, over a dozen parameters need to be set. Most were identified in the text, and in some cases, their values were specified in the discussion. The complete list is given in Appendix A. In arriving at the stated values, hundreds of runs were

conducted on several instances in an effort to strike a balance between runtime and solution quality.

Results for benchmark PDP data sets

The standard data sets used to compare heuristics for the PDP with time windows are those of Li and Lim (2001). They were derived from Solomon's (1987) VRPTW data sets and contain between 50 and 200 customer requests. We investigated the 50 customer instances which are divided into six groups denoted by LC1 (9 instances), LC2 (8 instances), LR1 (12 instances), LR2 (11 instances), LRC1 (8 instances) and LRC2 (8 instances). To put the comparisons on an equal footing, we set the number of vehicles available to the number of vehicles required in the best known solutions, which are given in Appendix B, and tried to minimize the total distance traveled. In each run, the number of Phase I iterations was fixed at 400, which took approximately 100 seconds, and the same parameter values were used, except for the Phase II iteration frequency and the removal-insertion parameter μ, which we varied. Based on our initial testing, no overall improvement was realized when Phase I was allowed to run beyond 400 iterations so we settled on that number. To compare performance of different settings under the same runtimes, the maximum total time allotted for Phase II was 600 seconds. As we shall see, for most of the Li & Lim instances, this value was large enough to ensure that the best known solutions were found.

In the first set of runs, our goal was to evaluate the performance of ALNS under different insertion-removal parameter values of μ. Since ALNS is essentially Phase II, we only ran it once on the best solution found during the 400 Phase I iterations. That is, we set the Phase II iteration frequency to 1/400. The GRASP was run four times on each instance––the first three times with the removal-insertion parameter μ fixed at 5, 10 and 15, respectively, and the fourth time with μ drawn reactively from the set {5, 6,…,15} at each Phase II iteration. Table 2.5 summarizes the results by data set. The first level column headings indicate the value of μ. For each data set, TB is the average time in seconds to the best solution and Gap is the percentage difference between the total distance (TD) found by GRASP and the best known solution (TD*) averaged over all instances in a particular category; that is Gap = 100×(TD – TD*)/TD* averaged over all instances. For a given data set, UI is the number of instances with unsatisfied customer requests in the GRASP solution / total number of instances in the date set. The bottom row in the table

presents the weighted average of the results for all data sets, where the weights are the number of instances in the corresponding data set. The total time allotted for each instances was approximately 700 seconds (= 100 sec for Phase I + 600 sec for Phase II).

Table 2.5. Results for Li & Lim data sets under different settings of μ

Data set		$\mu = 5$			$\mu = 10$			$\mu = 15$			Reactive μ		
Name	# test cases	TB (sec)	Gap (%)	UI	TB	Gap (%)	UI	TB (sec)	Gap (%)	UI	TB (sec)	Gap (%)	UI
LC1	9	114.76	2.40	0.00	102.48	0.57	0.00	112.35	0.22	0.00	72.22	0.42	0.00
LC2	8	123.71	2.70	0.00	113.50	0.00	0.00	107.67	0.00	0.00	129.42	0.00	0.00
LR1	12	111.19	0.04	0.33	166.97	0.06	0.08	209.33	0.00	0.08	172.47	0.05	0.00
LR2	11	641.25	4.21	0.18	573.92	0.00	0.09	616.66	0.08	0.00	685.57	0.00	0.09
LRC1	8	137.25	0.00	0.00	89.05	0.00	0.00	129.06	0.00	0.00	113.97	0.00	0.00
LRC2	8	446.63	8.34	0.13	431.21	0.00	0.00	430.25	1.06	0.00	415.79	0.06	0.00
Average		269.31	2.80	0.13	255.52	0.10	0.04	279.33	0.20	0.02	277.40	0.09	0.02

The results in Table 2.5 indicate that for the three fixed values of μ none is superior on all measures but the two larger values dominate $\mu = 5$ with respect to Gap and UI. The implication is that the latter value is too small to provide a diversified search. When $\mu = 15$ there is one instance with unsatisfied demand while the Gap is worse than the settings with $\mu = 10$ or reactive μ. When $\mu = 10$ the Gap is minimum with respect to the other fixed cases but there are two instances where not all customers' requests are satisfied. The reactive μ setting has the smallest Gap and like the case with $\mu = 15$, there is only one instance with unsatisfied demand. In all, the reactive approach works best by providing an effective balance between intensification and diversification of the local search.

In the second set of tests, our goal was to determine the frequency with which to run Phase II. Table 2.6 summaries the GRASP results for different combinations of Phase II iteration frequencies and corresponding runtime limits. For each instance, we experimented with the following four Phase II iteration frequencies in an effort to isolate the best setting: 1/400, 1/200, 1/100 and 1/40. To ensure a common means of comparison, we set the time limit for each run to

be 600, 300, 150 and 60 seconds, respectively, so the total time for Phase II was 600 seconds. In all runs, the removal-insertion parameter μ was determined reactively.

The first level column headings in Table 2.6 give the Phase II frequency and time limit combination for each setting. The output measures TB, Gap and UI are the same as in Table 2.5. Again, the total time allowed for the computations was 700 seconds per instance.

Table 2.6. Results for Li & Lim data sets under different Phase II iteration frequencies

Data set		Setting 1 (1/400, 600)			Setting 2 (1/200, 300)			Setting 3 (1/100, 150)			Setting 4 (1/40, 60)		
Name	# test cases	TB (sec)	Gap (%)	UI	TB	Gap (%)	UI	TB (sec)	Gap (%)	UI	TB (sec)	Gap (%)	UI
LC1	9	72.22	0.42	0.00	104.44	0.32	0.00	183.09	0.08	0.00	154.84	0.70	0.00
LC2	8	129.42	0.00	0.00	173.10	0.00	0.00	131.09	0.00	0.00	190.87	0.00	0.00
LR1	12	172.47	0.05	0.00	170.18	0.04	0.00	262.86	0.02	0.00	218.58	0.00	0.08
LR2	11	685.57	0.00	0.09	384.78	0.00	0.09	740.20	1.03	0.18	452.44	1.06	0.18
LRC1	8	113.97	0.00	0.00	104.30	0.00	0.00	261.27	0.00	0.00	300.90	0.00	0.00
LRC2	8	415.79	0.06	0.00	336.15	0.13	0.00	365.31	5.12	0.00	222.02	0.06	0.00
Avg.		277.40	0.09	0.02	216.48	0.08	0.02	339.39	0.95	0.04	262.57	0.33	0.05

For setting 1, Phase II is only executed once. For settings 2, 3 and 4, the Phase II is performed 2, 4 and 10 times, respectively. From the table, we see that setting 2 gives the best overall results with setting 1 a close second. One interesting observation is that for a given data set, if the Phase II time limit in a setting is greater than the TB result for setting 1, then the performance of that setting is generally no worse than that of setting 1. For example, for the LC1 data set, TB = 72.22 seconds for setting 1 which is less than the Phase II time limits of settings 2 and 3. The associated Gaps are 0.32% and 0.08%, respectively, and both are smaller than that of setting 1 which is 0.42%. For the LR1 data set, TB = 172.47 seconds for setting 1 which is less than the Phase II time limit of setting 2 and the associated Gap (0.04%) is smaller than that of setting 1 (0.05%). Similarly, the average TB for all the Li & Lim data sets under setting 1 is 277.40 which is less than the Phase II time limit of setting 2, and the average Gap of setting 2 (0.08%) for all data sets is smaller than that of setting 1 (0.09%).

These results were somewhat expected. If the Phase II time limit is greater than the average time to find the best solutions using setting 1, then it is likely that solutions of similar quality can be uncovered during each Phase II iteration. Under these conditions, the chances of finding superior solutions increase when Phase II is executed multiple times. In contrast, if the Phase II time limit is less than the TB for setting 1, then the GRASP may not be able to improve upon setting 1 solutions even when multiple Phase II runs are conducted. This was the case for settings 3 and 4 whose results were inferior to those obtained under setting 1.

Overall, the performance of the GRASP is comparable to the best heuristics in the literature for the PDP in terms of runtime and solution quality. Currently, the heuristic that gives the best results on the Li & Lim data sets outside of our work is the ALNS proposed by Ropke and Pisinger (2006). They report an average gap of 0.19%, which was obtained in 49 seconds on average. We reduced this gap to 0.08% under setting 2 but required an average of 216 seconds to do so. This is in line with their computations since Phase I takes about 100 seconds and our Phase II heuristic is an extension of ALNS. It is worth noting that in their analysis Ropke and Pisinger used the minimum number of vehicles reported by Li and Lim rather than the number associated with the best known solutions for distance traveled. In several instances, additional vehicles lead to shorter distances. The implication for our analysis is that lower Gap and UI values may be achieved by the GRASP than reported if the minimum number of vehicles reported by Li and Lim were used in our tests.

Without multiple starting points (i.e., without more than a single Phase I solution), we observed that ALNS may get trapped at local minima and fail to find good solutions. The ideal of reactively setting the removal-insertion parameter μ rather than using a fixed value helps to overcome this shortcoming. Of course, the longer Phase II is allowed to run regardless of how many times it is executed, the better the results are likely to be. If it is necessary to impose an overall limit on the computations, though, one needs to find a balance between the frequency with which Phase II is called and the time limit for each run. Multiple Phase II runs with different starting solutions help to diversify the search while longer Phase II runs ensure a more extensive exploration of local neighborhoods.

Results for randomly generated PDPT data sets

To evaluate the performance of the GRASP on instances with transshipments, we created five data sets with 25 requests and 1 transshipment location each using the procedure described in "data set generation" section of Section 2.5.2. The data sets are labeled pdpt1,…,pdpt5 and contain 10 instances each, all having the same optimal solution, which are given in Appendix C.

As with the Li & Lim data sets, two types of experiments were performed. Similarly, based on initial testing, the maximum number of Phase I iterations was set at 120 and the total amount of time allowed for Phase II was limited to 1200 seconds. These increases reflect the increased difficulty in finding good solutions when transshipment is permitted. Phase I required roughly 200 seconds apiece for each instance. In the first set of experiments our goal was to evaluate ALNS-Phase II under different removal-insertion parameter settings. Again the GRASP was run four times on each instance––the first three times with the removal-insertion parameter μ fixed at 5, 12 and 20, respectively, and the fourth time with μ drawn reactively from the set {5, 6,…,20}. Note that larger values of μ worked better for the pdpt data sets. In the second set of experiments we compared GRASP performance under different frequencies with which Phase II is called. Again the computations were performed under four settings for various Phase II frequencies and time limits. These included (1/120, 1200), (1/60, 600), (1/30, 300), (1/12, 120).

Table 2.7 reports the results for different values of μ. As expected, it was harder to find good solutions to the pdpt data sets than to the Li & Lim data sets even though the former contain 25 fewer requests per instance and were allowed to run for a longer period of time. Compared to the results in Table 2.5, the GRASP solutions now evidence larger gaps and indicate more instances with unsatisfied customer demand. This is attributed to the fact that the solution space is greatly expanded when the transshipment option is available. Finally, we again observe that it is better to adjust the removal-insertion parameter reactively than to use a fixed value.

Table 2.8 summaries the results for different Phase II execution frequencies. Setting 1, which only allows for a single Phase II run, proved to be inferior on measures of time and unsatisfied demand when compared to the three other settings, but superior with respect to the Gap. In the latter cases where Phase II is run multiple times, the UI results are invariably better. In part, this indicates that the solution space for the pdpt instances contains many local minima

and that ALNS can easily get trapped in one of them. When this happens, an average of 20% of the requests remain uncovered.

Because setting 1 produced the smallest gaps on average and had the largest time limit for the Phase II computations, it is reasonable to conclude that greater Phase II time limits are needed for settings with multiple Phase II runs to intensify the local search. Overall, the GRASP performed well and appears to be robust, at least for the data set examined. Under all setting, it was able to find a solution that satisfied all customer requests with a total travel distance no greater than 1% of the optimum for 80% to 88% of the instances (the values of UI ranged from 12% to 20%).

Table 2.7. Results for pdpt data sets under different settings of μ

Data set		$\mu = 5$			$\mu = 12$			$\mu = 20$			Reactive μ		
Name	#test cases	TB (sec)	Gap (%)	UI	TB	Gap (%)	UI	TB (sec)	Gap (%)	UI	TB (sec)	Gap (%)	UI
pdpt1	10	396.38	0.00	0.60	619.58	0.00	0.80	742.53	0.00	0.60	494.74	0.00	0.70
pdpt2	10	423.70	1.74	0.00	335.23	1.72	0.00	634.56	0.70	0.00	489.29	0.79	0.00
pdpt3	10	362.05	4.61	0.50	459.72	0.20	0.60	772.08	0.53	0.50	565.03	0.09	0.10
pdpt4	10	330.36	0.00	0.00	304.07	0.00	0.00	566.95	0.00	0.00	365.20	0.00	0.00
pdpt5	10	501.97	0.00	0.00	453.70	0.00	0.10	189.16	0.00	0.10	470.75	0.00	0.20
Avg.		402.89	1.27	0.22	434.46	0.38	0.30	581.05	0.25	0.24	477.00	0.18	0.20

Table 2.8. Results for pdpt data sets under different Phase II iteration frequencies

Data set		Setting 1 (1/120, 1200)			Setting 2 (1/60, 600)			Setting 3 (1/30, 300)			Setting 4 (1/12, 120)		
Name	# test cases	TB (sec)	Gap (%)	UI	TB	Gap (%)	UI	TB (sec)	Gap (%)	UI	TB (sec)	Gap (%)	UI
pdpt1	10	494.74	0.00	0.70	592.36	0.10	0.40	513.40	0.02	0.40	408.30	0.27	0.30
pdpt2	10	489.29	0.79	0.00	380.39	0.07	0.00	464.36	1.02	0.00	255.51	0.07	0.00
pdpt3	10	565.03	0.09	0.10	613.45	2.58	0.20	514.96	0.49	0.60	483.39	3.22	0.20
pdpt4	10	365.20	0.00	0.00	326.79	0.00	0.00	179.62	0.00	0.00	55.42	0.00	0.00
pdpt5	10	470.75	0.00	0.20	367.28	0.00	0.10	238.49	0.00	0.10	274.54	0.00	0.10
Avg.		477.00	0.18	0.20	456.05	0.55	0.14	382.17	0.31	0.22	295.43	0.71	0.12

Based on our analysis of the pdp and pdpt data sets, the following two-step procedure appears to be an effective way of tuning the GRASP for a given set of data. First, perform a preliminary run for each instance, executing Phase II only once. The goal here is to obtain an indication of the average time required to arrive at the best solution; call it τ^{avg} and set the total time allowed for Phase II to τ^{avg}. Second, set the number of times Phase II is to be executed to the maximum allowable runtime (defined by the user) divided by τ^{avg}. The resultant values should ensure that good solutions are found during local search without sacrificing the need to diversify the exploration of the solution space.

Results for cache implementation

We implemented cache structures to take advantage of prior insertion evaluations made as the GRASP traverses each of its two phases. One potential problem that may arise in this approach is that as more solutions are evaluated, the amount of data stored in cache grows linearly, implying that at some point it may take longer to search the cache then to directly perform the evaluations. Nevertheless, we never encountered this situation so we don't believe it is a concern, at least for the data sets tested.

While ALNS works with very large neighborhoods, it searches over far fewer neighborhoods than other heuristics such as tabu search, so the cache structures remain of manageable size. As the iterations progress the number of feasible insertions increase as does the time spent to access cache. For future data sets, if cache structures grow to the point that it takes

longer to search the cache then to perform the evaluations, we can clear the cache structures and start over.

Figure 2.16 provides a comparison for one data set with and without cache. We can see that the cached run gives superior performance.

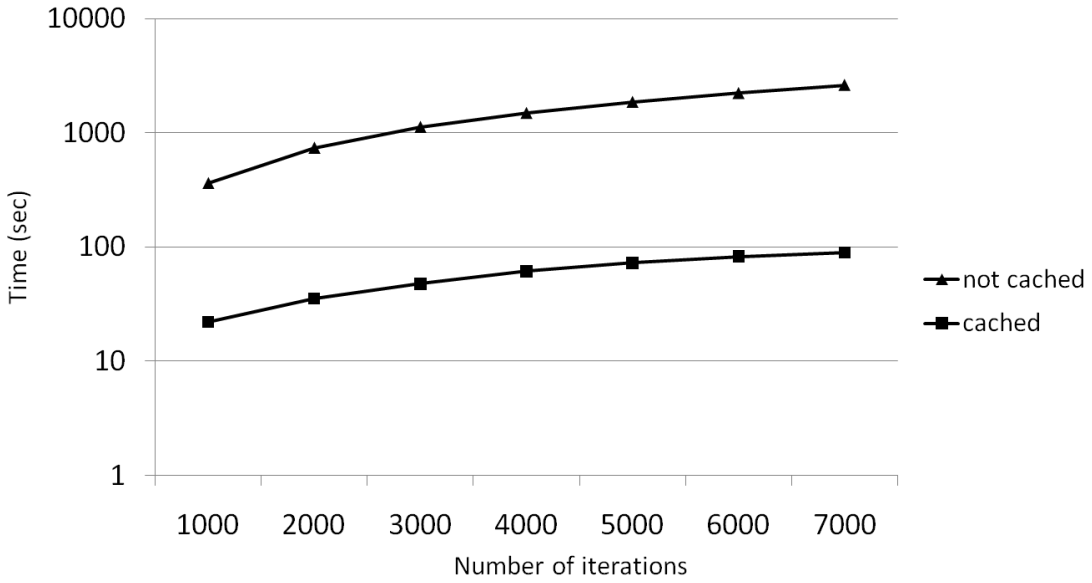

Figure 2.16. Performance comparison with and without cache

2.7 CONCLUSIONS

In this chapter, we proposed a unique way to present pickup and delivery problem with transshipment. We present a new MIP formulation for the PDPT problem and propose corresponding master problem and subproblem formulations for the branch and price method. We adapted the labeling algorithms to PDPT for solving the subproblem. The computation results show much better bound can be obtained by column generation comparing to the LP bound. The proposed branch and price can solve test cases with up to 6 customer requests to optimality in 2 hours, but has difficulty on larger test instances. To improve the performance of the branch and price methods by introducing valid inequalities to tighten up the relaxed master problem is a potential topic worth investigating.

To solve real life sized problem efficiently, we extend basic insertion heuristic used for most heuristic methods for pick and delivery problem to consider transshipment options. Based

on the representation of problem and extension of the basic insertion heuristic, most heuristic methods for pickup and delivery problem with time window can be easily adapted to solve the pickup and delivery problem with transshipment. We developed a GRASP and applied reactive ALNS as the local search heuristic to solve the PDPT, introduced insertion heuristics specialized for transshipment consideration. We also proposed methods to construct problem set for PDPT. The testing results show the heuristic can efficiently find near optimal solution in most test cases. The heuristic is designed to solve the PDPT which is a generalization of the PDP without transshipment, the test results also showed that it's competitive with known heuristics on the PDP. We are currently in the process of finding efficient column generation algorithm to solve PDPT based on same graph representation of the problem. Further research on heuristics can be done to solve larger PDPT test cases with more customer requests and transshipment locations.

Chapter 3. The Heterogeneous Pickup and Delivery Problem with Configurable Vehicle Capacity

3.1 INTRODUCTION

Although the pickup and delivery problem has been well studied in recent decades, real applications give rise to many variations that still challenge to the research community. The basic problem is defined by a fleet of vehicles available at one or more depots and a group of shipment requests at various locations scattered throughout a geographic area that need to be picked up and delivered by the vehicles within certain time windows. When there is more than one type of vehicle, the fleet is said to be nonhomogeneous (Xu et al. 2003). The shipment requests may contain one or more types of items (Parragh 2011), each with different capacity requirements, and each vehicle type may be able to handle different amounts of a particular type of request, depending on how the former is configured. In some cases, the capacity configuration of a vehicle can be changed to satisfy different shipment demand levels, as in this chapter. The objective of the PDP is typically to minimize some combination of ownership, mileage and travel time costs. We focus our attention on a PDP with a single depot, nonhomogeneous vehicle fleet, multiple shipment types, and configurable vehicle capacities.

Our interest in this problem was inspired by a real application associated with daily route planning for a Program of All-Inclusive Care for the Elderly (PACE) organization. The situation they face can be described as follows. On a daily basis, a set of participants must be picked up at their place of residence within a preferred time window and transported to either an activity center for socializing, or to a secondary facility such as a diabetes clinic or doctor's office for medical treatment. Later that day they are transported back to their residence at a requested time when possible. Most participants are ambulatory and simply need a seat in a van, the type of vehicle in the fleet, while others are wheelchair or walker bound and require more space. To accommodate a large mix of clients, PACE owns several different types of vans whose interiors can be readily modified to meet the transportation needs of their client. For example, seats on the vans can be raised or removed to allow for wheelchairs. Our objective is to first minimize the operational costs of providing service and then to minimize the number of vehicles used each day. A third objective is to minimize the average ride time of the participants.

The goal of this component of the dissertation, as suggested by the management of the PACE organization, is to develop a solution method to evaluate current practices and to enable the analysis of solutions under different what-if scenarios. For this purpose, we develop a multi-start adaptive large neighborhood search procedure that produced plans 30% to 40% less costly than current practice. One of the what-if scenario is to improve client satisfaction by allowing them to select a time to be picked up or delivered from predefined time slots to best fit their schedules. Under this setting, the time windows are more restricted, make an exact algorithm possible for instances of practical size. We propose a new solution algorithm using branch and price and cut for this purpose.

The main contributions of this study are as follows. First we present a new model for an unexamined variant of the PDP. Second, we develop a multi-start, adaptive large neighborhood search (MSALNS) algorithm to find solutions. The algorithm contains two phases. In phase I phase, a set of initial solutions is created with a greedy, randomized construction procedure. A promising subset of those solutions is then selected for further examination by solving a max diversity problem. In phase II, each candidate carried over from the first phase is improved with ALNS using reinsertion heuristics to intensify the search for local optima. The vehicle type assignments are made dynamically in both phases by solving transportation problems. Third is the development of new solution algorithm using branch and price and cut (B&P&C). A labeling algorithm is introduced to solve the pricing subproblem as an elementary shortest path problem. Efficient dominance conditions are proposed to speed up runtimes, and subset-row inequalities are used to strengthen the lower bound obtained by column generation. The fourth contribution is the introduction of benchmark data sets that can be used by the research community to further study the HPDP and to design more efficient solution techniques.

In the next section, we review the related literature. In Section 3.3, we outline the operations of the company that motivated the research and discuss its major characteristics. The MIP model is presented in Section 3.4 followed by descriptions of solution procedures. The multi-start ALNS heuristic, the B&P&C procedure and its implementation are introduced in Sections 3.6. Computational results for both solution methods are highlighted in Section 3.7 for benchmark problem instances as well as randomly generated instances. We conclude with an assessment of the methodologies and suggestions for future research.

3.2 LITERATURE REVIEW

Surveys for PDP can be found by Savelsbergh et al. (1995), Berbeglia et al. (2007) and Parragh et al. (2008). One well-studied PDP is the dial-a-ride problem (DARP) dealing with passenger transportation (Cordeau and Laporte 2003). Many researchers have developed state-of-the-art metaheuristics to solve routing problems based on GRASP, tabu search, simulated annealing, and neighborhood search and then extended them to find solutions to the PDP and DARP. For example, Qu and Bard (2012) developed a GRASP using adaptive large neighborhood search as local search algorithm for the PDP with transshipment. That is, they allowed shipments to be transferred among vehicles to better utilize fleet capacity. For the PDP with time windows, Nanry and Barnes (2000) proposed a reactive tabu search, Li and Lim (2001) used simulated annealing with tabu search, and Ropke and Pisinger (2006) developed an adaptive large neighborhood search heuristic. Parragh (2010) applied variable neighborhood search to the DARP.

An important variant related to our work concerns the heterogeneous VRP (HVRP) in which vehicles admitting different capacities and costs perform the distribution activities. If the number of available vehicles is unlimited, the HVRP along with the HPDP is termed a vehicle fleet mix problem (VFM). A comprehensive survey of the HVRP literature is provided by Baldacci et al. (2008) and Hoff et al. (2010). We now discuss some of the more recent work.

Due to the complexity of the problems, previous efforts to solve the HVRP and HPDP have mainly focused on heuristics. Dell'Amico et al. (2007) developed a two-phase heuristic for VFM with time windows. In the first phase, a solution is constructed using an insertion-based approach. In the second phase, the solution is improved by a ruin and recreate paradigm, where the incumbent is destroyed by removing some customers from routes, and then added back with a rebuilding procedure. The new solution is either accepted or rejected depending on the decision rule in force. If the new solution is accepted, it becomes the incumbent. The process is repeated until some stopping criterion is met. Testing was done on benchmark instances derived from the VRP instances created by Solomon (1987). The computational results showed improved performance compared to existing heuristics for the same problem.

Lee et al. (2008) used a modified sweep heuristic to construct feasible solutions to the VFM and then tabu search to converge to local optima. For each new solution found during an

iteration, a giant tour is created based on routes in the current solution. A set partitioning problem is then solved to optimally assign the vehicles to the routes constructed from the giant tour. The approach was tested on benchmark data sets with up to 100 customers and was able to improve on a few of the best known solutions.

Taillard (1999) proposed a heuristic for both the VFM and HVRP. First, a set of routes was constructed by solving a homogeneous VRP for each vehicle type with tabu search. Next, a set partitioning problem was solved to assign a vehicle to each route. Each customer was visited exactly once in the corresponding solution. On average, the heuristic produced better results than the ones published by Golden et al. (1984) for VFM instances with up to 100 customers. In the second phase of the study, VFM instances were used to generate HVRP instances by limiting the size of the fleets. The heuristic produced good solutions averaging less than 4% above the lower bound obtained by solving the VFM instances with free fleet composition. In most cases, total runtimes were less than one hour on a 50 MHz Sun Sparc workstation.

Prins (2002) applied a merge heuristic to the HVRP to obtain initial solutions, which were also improved by tabu search. He addressed a problem similar to ours where the number of vehicles of each type was fixed and multiple trips were allowed by each vehicle. The heuristic was tested using data from a French manufacture of furniture with 775 stores. It was estimated that over $2 million could be saved annually by applying the heuristic rather than the manual procedure used by the dispatchers.

Xu et al. (2003) investigated an HPDP encountered in real-world logistic operations. The feasible region was defined by nested precedence constraints, driver rules enforced by the U.S. Department of Transportation, and shipment and vehicle compatibility constraints. Using a set partitioning model as the master problem, they developed a heuristic pricing scheme to generate columns. Integer solutions were obtained by solving the master problem as an integer program at the root node; i.e., no branching was performed. The procedure was implemented in C on a PC with Pentium III 450MHZ processor, and relied on CPLEX to solve the master problem at all stages. Due to commercial confidentiality, testing was done on two sets of randomly generated instances instead of real data. For the medium size instances with up to 200 requests, the lower bound was obtained from the master problem and the subproblem was solved by dynamic programming. The optimality gaps were mostly less than 5% and runtimes were all less than 5

minutes. For the large size instances with 300 to 500 requests, finding feasible solutions took up to 7 hours but no lower bounds or optimality gaps could be obtained in a reasonable amount of time.

In terms of exact solution methods, which, for the PDP, have mostly centered on homogeneous fleets and a single demand type. Dumas et al. (1991) developed a branch-and-price scheme and proposed a dynamic programming labeling algorithm to solve the subproblem – a constrained shortest path problem. They applied dominance tests to reduce the size of the state space, and introduced ways to preprocess the input data to tighten the time windows and eliminate infeasible arcs in the underlying network. This further reduced the set of feasible routes. Testing showed that the labeling algorithm was efficient when the maximum number of requests served by a vehicle was 5.

Sol and Savelsbergh (1994) proposed a similar branch-and-price scheme but solved the constrained shortest path problem with a construction and improvement heuristic. For branching, an assignment-based rule was used instead of a route selection-based rule as in Dumas et al. (1991). Because the subproblems were not solved exactly, the approach must be considered a heuristic. Its performance was evaluated using randomly generated test cases with 30 requests. Each was first solved to optimality with an exact method to establish a frame of reference. The testing showed that the three approximation schemes yielded high quality solutions within a few percentage points of the optima but in a fraction of the time.

Ruland and Rodin (1997) explored the polyhedral structure of one formulation of the DARP and developed four classes of valid inequalities based on subtour elimination and precedence constraints. Their branch-and-cut algorithm was implemented on a Sun SparcServer 670 and could optimally solve randomly generated instances with up to 15 requests within 30 minutes. Ropke et al. (2007) proposed more sophisticated branch-and-cut algorithms based on two new 2-index formulations. Their implementation included two new types of valid inequalities: strengthened capacity constraints and fork constraints. Testing was done on a PC with a 2.4GHZ AMD processer and produced optimal solutions for instances with up to 8 vehicles and 96 requests.

Subsequently, Ropke and Cordeau (2009) proposed a branch-and-price-and-cut algorithm for the PDP with time windows. Two pricing subproblems were considered, the first being a

shortest path problem and the second an elementary shortest path problem. In addition, several classes of inequalities were used, including rounded capacities inequalities, precedence constraints, and two-path cuts common to vehicle routing problems (e.g., see Bard et al. 2002). The algorithms were implemented in C++ on an AMD Opteron 250 computer with a 2.4GHz processor. The test results showed that regardless of subproblem, the solutions were comparable. When the time windows were tight, large instances with up to 500 requests were solvable but otherwise, it was difficult if not impossible to even get feasible solutions.

Taking a dual approach, Baldacci et al. (2011) developed an exact algorithm for the PDP with time windows based on a set partition formulation. They introduced a bounding procedure that used two dual ascent heuristics to find lower bounds on the LP relaxation. Coupling those lower bounds with known upper bounds, they were able to substantially reduce the number of variables considered in the formulation, and directly solve moderate sized instances with commercial software. Branch and price and cut was used for the larger instances. Testing was done on two benchmark sets, one from Ropke and Cordeau (2009) with up to 75 requests and the other from Li and Lim (2001) with up to 100 requests. Several optimal solutions were obtained for a few open cases.

Parragh (2011) studied a DARP with different types of users and vehicles, as in our problem, but assumed that the vehicle capacities were fixed. To find solutions, she proposed a 2-index and a 3-index formulation and implemented branch-and-cut algorithms for each. Upper bounds were provided by an extended version of a DARP variable neighborhood search heuristic. The algorithms were implemented in C++ and run on a 3.2GHZ Pentium D computer using CPLEX11.0 with Concert Technology as the solver. The PDP instances from Cordeau (2006) containing 2 – 4 vehicles and 16 – 48 requests were modified to include heterogeneous users and vehicles. In a comparative study, it was shown that the branch-and-cut algorithm applied to the 2-index formulation was able to solve most instances with up to 40 requests to optimality and outperformed the 3-index formulation. For the larger instances, feasible solutions were obtained with the heuristic.

Although extensive research exists on the HVRP and HPDP, only a few studies have considered different types of customer demand. To the best of our knowledge, there is no

published work on the HPDP that addresses configurable vehicle capacity, the novel component of our work.

3.3 Description of Daily Operations at PACE

As the population ages, a growing number of seniors face the decision of whether to seek more supportive living arrangements. At one extreme are nursing homes that provide custodial care under the supervision of physicians, nurses and other medical professionals. A less dependent alternative are assisted living facilities where the residents get less personal help. Nevertheless, a vast majority of seniors still prefer to live in their own homes where they have an emotional attachment even though they may not be fully capable of caring for themselves. To address this issue, PACE was established by Medicare to serve older adults and anyone over 55 living with disabilities. This program provides a cost-efficient alternative for people who otherwise need nursing home levels of care.

The mission of the roughly 82 PACE organizations in the U.S. is to help participants live in their community for as long as possible. To accomplish this, they focus on preventive care by helping participants with their medical-related errands such as picking up prescriptions, going to the doctor's office for checkups, and attending clinical sessions. Although everyone enrolled in PACE is eligible for nursing home care, only 7% have opted for it.

A typical day of a participant starts in the morning when he or she is picked up by a PACE van within an agreed upon time window, and is transported to either the activity center or to a secondary location. The daily agenda mainly includes social activities although it is possible to receive routine medical care on site if an appointment was previously scheduled. Other available services include meals, therapy, personal care and counseling. In some cases, participants may need transportation for off-site appointments at diagnostic centers, out-patient clinics, and doctors' offices. At the end of day, they are dropped off at their place of residence.

To satisfy these transportation requirements, each PACE organization owns a fleet of configurable vans. The interior of each van can be modified to accommodate a different number of passengers, depending on their level of mobility. For example, some participants may need a wheelchair while others only a cane or walker. Changing a vehicle's capacity is a matter of removing or lifting a bench or folding down a seat, and can be done in minimal time.

At virtually all PACE organizations, routes and schedules are constructs a few days in advance by hand by the transportation supervisor with the help of a database system that stores client information. This is a frustrating exercise and often produces unsatisfactory results. We believe that developing more efficient scheduling procedures will both lower operational costs and increase client satisfaction, two factors that are vital to the success of PACE. The average daily ride time to and from the activity center, the flexibility of the schedule for pickup and drop off, and the comfort level of the vans all affect the willingness of potential participants to join the program.

With this background in mind, we modeled the daily routing problem faced by each PACE organization as an HPDP with side constraints. The latter include time windows for pickup and drop off, simultaneous service requirements for those participants residing at the same location, and a need to accommodate equipment needed for individual mobility. The objectives are to minimize the overall cost of transportation services and to reduce the overall ride time of each participant.

3.4 PROBLEM DEFINITION

In general terms, the HPDP with configurable vehicle capacity can be formally modeled on a directed graph $G = (N, A)$. The node set $N = P \cup \{b, b'\}$, where P is the set of client requests and $\{b, b'\}$ represents the depot start and end nodes, and arc set A consists of all feasible links.

Each client request $c \in C$ is associated with a set of load requirements $\{L_{cd} : d \in D\}$ for each type of demand $d \in D$ to be picked up at node i_c and delivered to node j_c. In our problem, d can be a seat, a walker or a wheelchair. All vehicles are stationed at the depot where they start and end the day. The sequence of nodes visited by a vehicle including the depot nodes b and b' is called a route $r \in R$. For all $i \in N$, a time window $[a_i, b_i]$ is defined that specifies the earliest and latest time service can begin, regardless of node designation. If a vehicle arrives at location i prior to a_i, it has to wait until a_i to start service, which has a fixed duration of σ_i minutes. Each vehicle is categorized by a type $f \in F$; each vehicle type has a set Λ_f of configuration options that provides different combinations for the various categories of capacity types; i.e., option $\lambda \in \Lambda_f$ provides capacity $Q_{f\lambda k}$ for capacity type $k \in K$. Each capacity type can accommodate different levels of demand types; i.e., one unit of capacity type $k \in K$ can satisfy q_{kd} units of demand type

$d \in D$. The capacity of a vehicle cannot be exceeded along its route and the configuration option selected for a vehicle cannot be changed along its route.

Directed arcs exist between most nodes except in the following cases: there are no arcs from any node to b nor are there any arcs from b' to any other node; no client pickup node can connect to b' (all passengers must be dropped off before the vehicle returns to the depot node; when the drop off point is at the same location as the depot, they are represented with two different nodes); there are no arcs from b to client delivery nodes (no person can be dropped off before being picked up); and there are no arcs from a delivery node to the pickup node of the same client. An arc can be removed if no feasible route can be constructed that uses the arc; see Dumas et al. (1991) for further discussion of this matter.

Three interrelated objectives govern route construction as mentioned above. The first is to minimize the number of vehicles required to service all clients subject to capacity and time window constraints. The second is to minimize the total distance traveled, which is proportional to the variable cost of providing transportation. The third is to minimize shipment travel time on the vehicles. This is important in our case because the "shipment" is a passenger; shorter travel times mean greater passenger satisfaction. The relative importance of these objectives can be adjusted by scaling their cost coefficients. In our problem, we assume that travel cost and travel time satisfy the triangle inequality.

In the developments, we make use of the following notation, some of which has been previously defined.

Indices and sets

A	set of arcs
C	set of client requests; $c \in C$
D	set of demand types having different capacity requirements; $d \in D$
F	set of vehicle types; $f \in F$
K	set of capacity types; $k \in K$
Λ_f	set of vehicle configurations for vehicle type $f \in F$; $\lambda \in \Lambda_f$
N	set of nodes including depot nodes, pickup nodes, delivery nodes; $i, j \in N$
b, b'	indices for depot start, end node
i_c	pickup node index for request $c \in C$

j_c delivery node index for request $c \in C$

$NF(i)$ set of immediate successor nodes of node i; arc (i,j) exists for $j \in NF(i)$

$NB(i)$ set of immediate predecessor nodes to node i; arc (j,i) exists for $j \in NB(i)$

Parameters

n_f number of available vehicles of type $f \in F$

c_{fij} cost of traversing arc $(i,j) \in A$ including the fixed cost c_f for using vehicle when arc starts with depot start node b.

τ_c penalty for each minute that client $c \in C$ spends on a vehicle

T_{ij} travel time on arc $(i,j) \in A$ (includes service time σ_i)

L_{cd} load for client $c \in C$ for demand type $d \in D$

Δ_{id} load change at each node $i \in N$ for demand type $d \in D$; if $i = i_c$, then $\Delta_{id} = L_{cd}$; if $i = j_c$, then $\Delta_{id} = -L_{cd}$, otherwise $\Delta_{id} = 0$

$Q_{f\lambda k}$ capacity of vehicle type $f \in F$ for capacity type $k \in K$ under configuration $\lambda \in \Lambda_f$

q_{kd} number of units of demand type $d \in D$ that can be accommodated by one unit of capacity type $k \in K$

$[a_i, b_i]$ time window for node $i \in N$

T^{max} latest time any delivery (beginning of service) can occur

Decision variables

$x_{ij}^{f\lambda}$ (binary) 1 if vehicle of type f with configuration λ traverses arc (i,j), 0 otherwise

y_{ij}^{dk} amount of capacity type k assigned for demand type d on vehicle traversing arc (i,j)

z_{ij}^{d} amount of demand type d on a vehicle while traversing arc (i,j)

t_i time service starts at node i

Model

$$\text{Minimize} \sum_{f \in F} \sum_{\lambda \in \Lambda_f} \sum_{(i,j) \in A} c_{fij} x_{ij}^{f\lambda} + \sum_{c \in C} \tau_c \left(t_{j_c} - t_{i_c} \right) \tag{3.1a}$$

subject to

Vehicle flow balance

$$\sum_{j \in NF(b)} \sum_{\lambda \in \Lambda_f} x_{bj}^{f\lambda} \leq n_f, \ \forall \ f \in F \tag{3.1b}$$

$$\sum_{j \in NB(i)} x_{ji}^{f\lambda} - \sum_{j \in NF(i)} x_{ij}^{f\lambda} = 0, \ \forall \ i \in N \setminus \{b, b'\}, \ f \in F, \ \lambda \in \Lambda_f \tag{3.1c}$$

$$\sum_{j \in NF(b)} x_{bj}^{f\lambda} = \sum_{i \in NB(b')} x_{ib'}^{f\lambda}, \ \forall \ f \in F, \ \lambda \in \Lambda_f \tag{3.1d}$$

Client flow balance

$$\sum_{f \in F} \sum_{\lambda \in \Lambda_f} \sum_{j \in NF(i_c)} x_{i_c j}^{f\lambda} = 1, \ \forall \ c \in C \tag{3.1e}$$

$$\sum_{j \in NF(i_c)} x_{i_c j}^{f\lambda} = \sum_{i \in NB(j_c)} x_{ij_c}^{f\lambda}, \ \forall \ c \in C, \ f \in F, \ \lambda \in \Lambda_f \tag{3.1f}$$

$$\sum_{j \in NF(i)} z_{ij}^{d} - \sum_{j \in NB(i)} z_{ji}^{d} = \Delta_{id}, \ \forall \ i \in N, \ d \in D \tag{3.1g}$$

Vehicle capacity configuration

$$z_{ij}^{d} \leq \sum_{k \in K} q_{kd} y_{ij}^{dk}, \ \forall \ d \in D, \ (i, j) \in A \tag{3.1h}$$

$$\sum_{d \in D} y_{ij}^{dk} \leq \sum_{f \in F} \sum_{\lambda \in \Lambda_f} x_{ij}^{f\lambda} Q_{f\lambda k}, \ \forall \ k \in K, \ (i, j) \in A \tag{3.1i}$$

Time continuity

$$t_i + T_{ij} \leq t_j + \left(1 - \sum_{k \in K} x_{ij}^{k}\right) T^{max}, \ \forall \ (i, j) \in A \tag{3.1j}$$

$$a_i \leq t_i \leq b_i, \ \forall \ i \in N \tag{3.1k}$$

Variable definitions

$$x_{ij}^{f\lambda} \in \{0,1\}, \ \forall \ f \in F, \ \lambda \in \Lambda_f, \ (i, j) \in A; \ y_{ij}^{dk} \in Z_{+}^{1}, \ \forall \ d \in D, \ k \in K, \ (i, j) \in A$$

$$z_{ij}^{d} \geq 0, \ \forall \ d \in D, \ (i, j) \in A; \ t_i \geq 0, \ \forall \ i \in N \tag{3.1l}$$

The objective function (3.1a) is designed to minimize total weighted costs. The first term sums the costs associated with the arcs traversed by all vehicles in a solution, including the fixed cost c_f for using a subset of the fleet. If a type f vehicle leaves the depot and visits at least one client, then a cost of c_f is incurred. The second term is the cumulative penalty for the time that the clients spend on a vehicle. It is included to minimize the individual ride times when there are

multiple optimal solutions. All things being equal, it is preferred to pick up clients who reside near the depot towards the end of a route rather than near its origin.

Constraints (3.1b) restrict the number of times vehicles of type f with configuration λ can leave the depot to N_f, which would be the case if they were used for routes. Constraints (3.1c) and (3.1d) ensure that vehicle flow balance is maintained at each client node and at the depot, respectively; if a vehicle of type f with configuration λ leaves the depot it must return. Constraints (3.1e) - (3.1g) guarantee that flow is balanced for each request for service. Constraints (3.1e) ensure that each client is picked up by exactly one vehicle and (3.1f) ensure that the vehicle that picks up that client will also drop him off. Constraints (3.1f) also ensure that no configuration change for a vehicle along its route. Constraints (3.1g) enforce client flow balance on a vehicle at each node. If a vehicle visits a pickup node i_c for client c, then it will pick up the loads in the set $\{L_{cd} : d \in D\}$; some clients might have more than one type of demand, i.e., a couple must be picked up together (and so is treated as one client) and if one or both require a wheelchair, then that demand must be taken into account; if a vehicle visits a delivery node j_c for client c, then it will drop off the loads in the set $\{L_{cd} : d \in D\}$; otherwise, the load on the vehicle for each demand type remains unchanged.

The capacity restriction for each demand type $d \in D$ is enforced by constraints (3.1h) and (3.1i). In particular, constraints (3.1h) ensure that the quantity of flow on arc (i,j) associated with demand type d must be less than or equal to the total capacity available for the demand provided by all capacity types assigned to the demand type on the arc (i,j); constraints (3.1i) ensure that each capacity type k assigned to arc (i,j) is no more than the amount permitted by the selected configuration of the vehicle type serving that arc. It's noteworthy that we don't need index f on variables y_{ij}^{dk} and z_{ij}^{d} here. Because each customer node is only served by one vehicle based on constraints (3.1e) and (3.1c), we know that there is only one unique vehicle type f associated with an arc. So with arc index (i, j), it's enough to determine the vehicle type f associated with y_{ij}^{dk} and z_{ij}^{d} variables.

Constrains (3.1j) ensure that time is increasing between any two nodes connected by an arc. Time continuity has to be observed for arc (i,j) when a vehicle visits nodes i and j in succession; otherwise, the constraints are redundant. An added benefit provided by (3.1j) is that it prevents subtours from arising that don't start and end at the depot. Constraints (3.1k) ensure

that the time service begins at a node satisfies the time window restrictions. Finally, variable definitions are given in (3.1*l*).

Example 3.1. The following example illustrates how configurable capacity can be beneficial in cost reduction. Assume that there is a single vehicle with 2 benches that can provide 4 seats with space remaining for 1 wheelchair (and hence a fifth person). Alternatively, it is possible to lift one of the benches to accommodate an additional wheelchair, and therefore provide 2 seats and 2 wheelchair spaces. There are four clients who need transportation: c_1, c_2, c_3 and c_4. The locations of the depot, pickup and delivery points are as shown in Figure 3.1. The destinations of clients c_1, c_2 and c_4 coincide with the depot b and hence are not shown in the figure. Both clients c_3 and c_4 require a wheelchair. For simplicity, we assume that the objective is to serve all clients and minimize the total distance traveled by the vehicle.

In case of configurable capacity, the best route option is b, i_{c_1}, i_{c_3}, i_{c_4}, i_{c_2}, j_{c_3}, j_{c_1}, j_{c_2}, j_{c_4}, b', as indicated by the solid arrows in Figure 3.1, where $b' \equiv b$. However, if the vehicle has only a single configuration, i.e., 4 seats and accommodation for 1 wheelchair, then the best route is b, i_{c_1}, i_{c_3}, j_{c_3}, i_{c_4}, i_{c_2}, j_{c_1}, j_{c_2}, j_{c_4}, b', in which client c_3 has to be dropped off to free up space before picking up client c_4 who also requires the transportation of a wheelchair (see dashed arrows in the figure). Let $D(i, j)$ be the distance between nodes i and j. Assuming that $D(i_{c_2}, b') = D(i_{c_2}, j_{c_3}) + D(j_{c_3}, b')$, then the additional distance travelled is $D(i_{c_3}, j_{c_3}) + D(j_{c_3}, i_{c_4}) - D(i_{c_3}, i_{c_4})$. ■

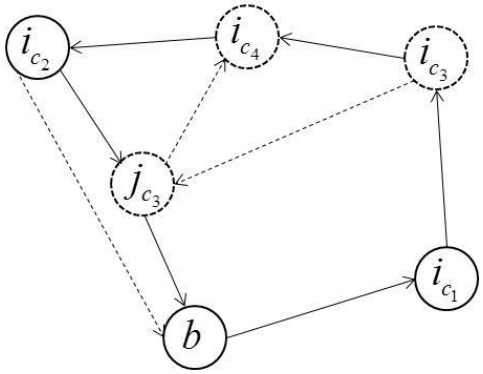

Figure 3.1. Graph for Example 3.1

Proposition 3.1. The PDP with configurable vehicle capacity is a relaxation of the PDP with fixed vehicle capacity when the vehicles are homogeneous and their capacity corresponds to one of the configurable options.

Proof. Any solution for the PDP with one of the fix capacity configurations is also valid for the PDP with configurable vehicle capacity. Example 3.1 shows that it is possible to achieve a cost reduction by taking advantage of a vehicle's configurability. ■

Proposition 3.2. A necessary condition for a vehicle with configurable capacity to provide a cost reduction with respect to a vehicle with a corresponding fixed capacity is that the total client demand exceeds the fixed vehicle capacity.

Proof. If the total client demand does not exceed the fix vehicle capacity, then any solution under the configurable vehicle capacity would also be valid for the fixed vehicle capacity. ■

Several attempts to solve model (3.1) with CPLEX for real instances with upwards of 100 participants met with limited success. The largest instances for which the optimum could be found within 2 hours contained 1 vehicle and 8 participants. These motivated the development of a metaheuristic which is described in the next section.

3.5 SOLUTION METHODOLOGY

3.5.1 Multi-start ALNS

Both exact algorithms and metaheuristics have been developed to solve general PDPs with time windows. Specialized exact algorithms such as branch and price and cut along with efficient lower bounding schemes have reliably solved benchmark instances with up to 100 customers, and less frequently with several hundred customers (e.g., see Ropke and Cordeau 2009, Baldacci et al. 2011). However, as Ropke and Cordeau have stated, loosely constrained instances remain challenging. For large-scale applications arising in practice, metaheuristics provide a good alternative. Researchers have had notable success in solving large instances of VRPs using two types of local search algorithms. Simulated annealing (Li and Lim 2001) and tabu search (Nanry and Barnes 2000) represent the first type in which "small" neighborhoods are explored at each step in an effort to improve the current solution. Large neighborhood search (LNS) is the second type, and greatly expands the topological landscape. At each step, a substantial part of the current solution is destroyed and then reconstructed, thus providing a much broader search.

When different types of neighborhoods are defined, we get what is often called *variable neighborhood search* (Hansen and Mladenovic 2001).

Recently, Ropke and Pisinger (2006) used LNS on VRPs and PDPs with time windows to get high quality solutions to instances with up to 500 customers. They extended the ideas proposed by Shaw (1998) to arrive at what they called ALNS. This approach begins with an initial solution and iteratively destroys and rebuilds it by randomly choosing and applying a number of quick neighborhood search heuristics. Associated with each heuristic is a weight that determines its selection probability. At each iteration, the new solution is either accepted or rejected and the heuristic selection weights are updated according to their performance. For the PDP, a solution is destroyed by removing a number of customers from the routes using tailor-made procedures, and then reinserting them back into perhaps different routes, depending on the logic of the insertion heuristics. Qu and Bard (2012) developed a GRASP that used ALNS in the improvement phase. Similarly, we propose here to use a two-phase heuristic that relies on ALNS in phase II but now with a reinsertion local search that serves as an intensification mechanism.

In particular, our algorithm, which we call multi-start ALNS or MSALNS, contains three major components: a randomized construction procedure, a randomized destruction procedure, and a local search centered on neighborhood reinsertion. In phase I, a predefined number N_0 of initial solutions are constructed from scratch. The best N_1 of them are feed to a max diversity problem, which filters the number down to N_2. In phase II, each solution in the N_2 solutions is improved by ALNS until a predefined runtime or iteration limit is reached. The resulting solution is then further improved by a reinsertion local neighborhood search (RILS). A flow diagram of the computations is depicted in Figure 3.2.

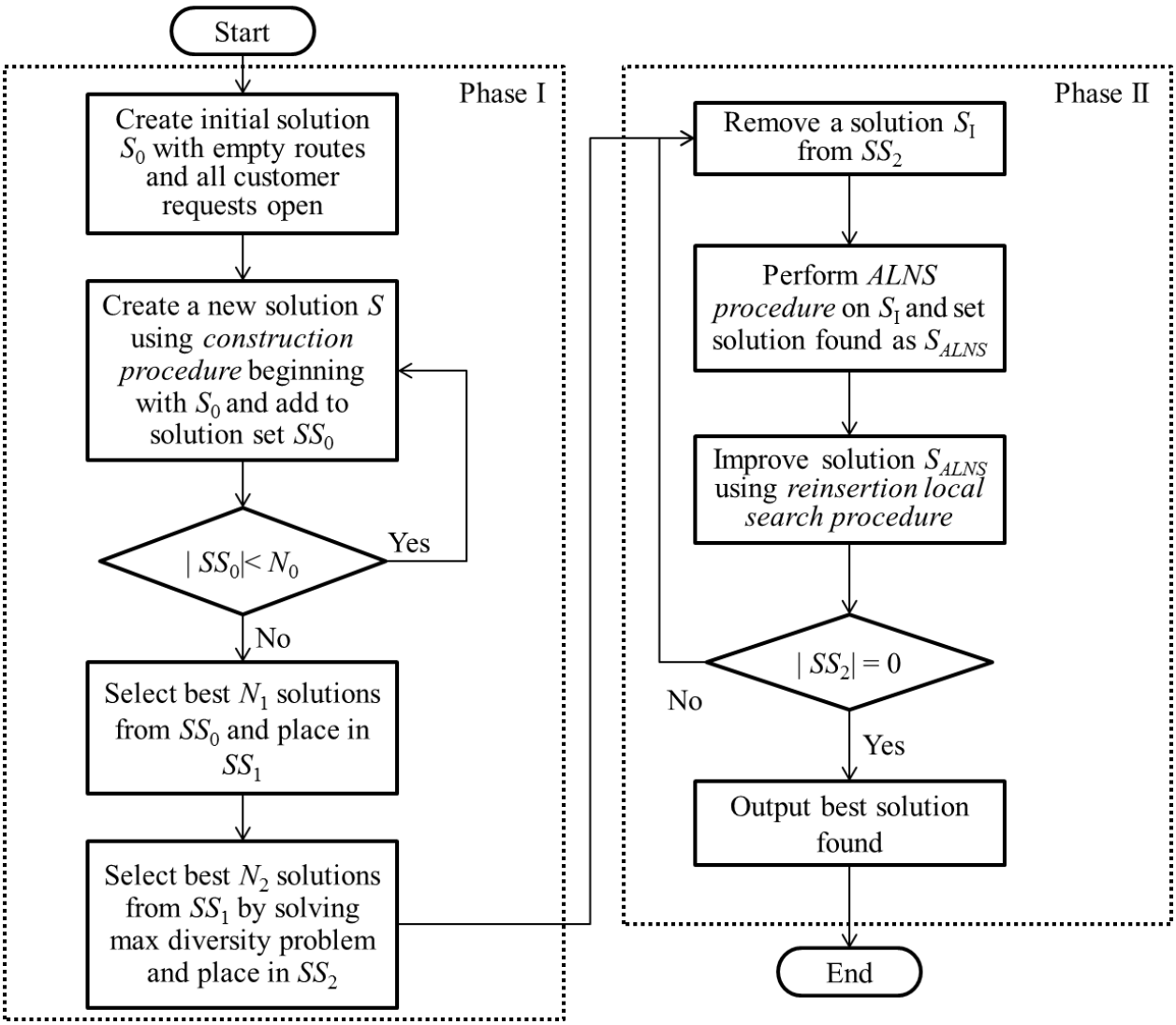

Figure 3.2. Flowchart for multi-start ALNS

Phase I

The main purpose of phase I is to find a set of high quality feasible solutions (SS_2) to be locally optimized in phase II. These solutions should not only have a low objective function value but should be widely distributed over the feasible region. A high degree of diversity improves the chances that the global optimum will be found during phase II. We start with a null solution S_0 where each vehicle is assigned an empty route that starts and ends at the depot, and all client requests are open. Initially the vehicle types are randomly assigned to the routes.

The construction procedure used to produce a set of initial solutions, SS_0, is similar to the one used by Qu and Bard (2012). Each solution is built iteratively by inserting client requests

into partial routes. At each step, a candidate list is formed by identifying the clients that would improve the objective function the most if inserted into a route. One client is then randomly selected from those at the top of the list (the size of the candidate list is an adaptive parameter) and inserted into a partial route. In our algorithm, though, instead of using a candidate list, we select the next client by applying one of several insertion heuristics that is randomly chosen at each iteration. The higher a customer request is on the list, the better the chance that it is inserted next. Given a selected client request and a partial solution, we evaluate all possible insertion positions and choose the one that adds the least cost to the objective function.

The pseudocode for the randomized construction procedure is shown in Figure 3.3. Starting with a null solution $S_0 = \varnothing$ and all yet-to-be routed clients in the set $U(S_0)$, the procedure inserts client requests iteratively. Given a partial solution S at some iteration, one client $c \in U(S)$ is selected from an ordered list OL of clients in $U(S)$ in accordance with one of the insertion heuristics. Letting p be a predefined number to gauge the random effect, we first generate a random number *rand* in [0, 1) and select the $\lfloor rand^{p} \mid OL \mid \rfloor^{\text{th}}$ element on OL to insert. If $p = 0$, then the client is randomly selected from OL. If $p = \infty$, then the first element in OL is always chosen. In the implementation, we set $p = 5$.

Inserting client c into some route r means determining the positions of nodes i_c and j_c. There are often many options, which are represented by the set IN_c. The best insertion point with respect to the partial solution S is defined as the position $\wedge_c$ that results in smallest increase in the cost realized so far; that is, $\wedge_c = \arg\min_{\Lambda \in IN_c} \{COST(S'(S, \wedge))\}$, where $COST$ is the objective function value defined in (1a) and $S'(S, \wedge)$ is the solution after insertion $\wedge$ is applied to S. This process is repeated until the set $U(S)$ is empty. In the next section, we describe in details how OL is determined for each insertion heuristic.

Procedure `Randomized_Construction`

Input: Initial solution S_0, a set of insertion heuristics $\{ih_1, \ldots, ih_{ni}\}$ and their corresponding weights $\{iw_1, \ldots, iw_{ni}\}$, a scalar p

Output: New solution S

Step 0: Set $S = S_0$;

Randomly select an insertion heuristic ih_i based on their weights;

Step 1: While ($|U(S)| > 0$) //there are client requests not satisfied

Create an ordered list OL from clients in $U(S)$ according to ih_i;

Generate random number $rand \in [0, 1)$;

Select the next customer $c = OL[\lfloor rand^{\,p} \mid OL \mid \rfloor]$;

Find best position $\wedge_c$ into which to insert c into routes in S;

If ($\wedge_c$ exists), then

Update S with $\wedge_c$ and remove c from $U(S)$ to get $S = S'(S, \wedge_c)$;

Else

Go to Step 2;

Step 2: Return S.

Figure 3.3. Pseudocode for randomized construction procedure

Insertion heuristics

Given a partial solution S with unrouted clients $U(S)$, the *greedy insertion heuristic* (ih_1) evaluates all $c \in U(S)$ one at a time to determine the best insertion position of each. The list OL is constructed in ascending order of their objective function values $COST(S'(S, \wedge_c))$.

Regret-k insertion heuristic (ih_2) selects the client $c \in U(S)$ that would produce the greatest regret if it were not inserted first. For a given c, let $\wedge_c^k$ be the k^{th} best insertion. The list OL is constructed in descending order of $COST(S'(S, \wedge_c^k)) - COST(S'(S, \wedge_c))$. In our implementation, $k = 2$.

Random insertion (ih_3) randomly decides the order in which requests are inserted.

Most constrained first insertion (ih_4) orders the requests based on descending order of a weighted function of the distance between supply and demand locations, time window periods and request load for each type of demands. The corresponding function is

$$Q_c = \beta_1 D_{i_c j_c} + \beta_2 / \left(\left| b_i - a_i \right| + \left| b_j - a_j \right| \right) + \sum_{d \in D} \beta_{3d} L_{cd} \tag{3.2}$$

where β_1, β_2 and β_{3d}, $d \in D$ are weights for each component in (2). This logic is motivated by the fact that if a request is hard to accommodate, we want to consider it first. That is, requests that require longer travel distances, have narrow service time windows, and have larger request loads are more difficult to fit into a route.

Maximum diversity problem

The randomized construction procedure is called N_0 times to build a set of initial solutions, SS_0. The solutions in SS_0 are sorted based on their objective function values and the best N_1 are selected to form the set SS_1. We then solve what we call a maximum diversity problem (MDP) to select N_2 solutions from SS_1 to get the set SS_2 of solutions that is passed to phase II. The MDP formulation is given below.

Indices and sets

S set of solutions

s, s' indices for solutions

Parameters

$d_{ss'}$ distance between two solutions s and s'

m number of solutions to be selected

Decision variables

ζ_s (binary) 1 if solution s is selected, 0 otherwise

$\nu_{ss'}$ (binary) 1 if solution s and s' are both selected, 0 otherwise

Model

$$\text{Maximize} \sum_{s \in S} \sum_{s' \in S} d_{ss'} \nu_{ss'} \tag{3.3a}$$

$$\text{subject to} \quad 2\nu_{ss'} \le \zeta_s + \zeta_{s'}, \ \forall s, s' \in S \tag{3.3b}$$

$$\sum_{s \in S} \zeta_s = m \tag{3.3c}$$

$$\zeta_s \in \{0,1\}, \forall s \in S\ ;\ \nu_{ss'} \in \{0,1\}, \forall s, s' \in S \tag{3.3d}$$

In our problem, $m = N_2$ and $S = SS_1$. The distance $d_{ss'}$ between two solutions s and s' is defined as $\sum_{c_1 \in C} \sum_{c_2 \in C} d_{ss'}^{c_1 c_2}$, where $d_{ss'}^{c_1 c_2}$ is calculated as follows. Given two customers c_1 and c_2, we initialize the distance between them, $d_{ss'}^{c_1 c_2}$, to be 0. According to the positions of customers c_1 and c_2 being served in routes of the solutions s and s', we have the following four cases: case I, c_1 and c_2 are served by different vehicles in both solutions; case II, c_1 and c_2 are on the same route in one of the solutions but not in the other; case III, they are on the same route in both solutions and the relative visiting orders of the pickup and delivery nodes of c_1 and c_2 differ, i.e., the visiting order in one solution is i_{c_1}, j_{c_1}, i_{c_2}, j_{c_2} and i_{c_1}, i_{c_2}, j_{c_2}, j_{c_1} in the other; case IV, they are served by the same vehicle in the same visiting order in both solutions.

In the construction of $d_{ss'}^{c_1 c_2}$, the distance between the two customers is implicitly defined to be the minimum number of reinsertion moves necessary to convert s to s'. In case II, at least one reinsertion move must be performed on one of the customers to transform the first solution into the second. The same is true for case III. In all other cases, either the two customers are served by different vehicles in both solutions or they are served by the same vehicle in the same visiting order in both solutions. As such, we say that no "necessary" reinsertion moves are required for these customers to convert one solution to the other because we can reorder the other customers on the vehicles to achieve the same absolute positions for c_1 and c_2 in both solutions. Let δ_1 and δ_2 be two nonnegative parameters, in case II, we increment $d_{ss'}^{c_1 c_2}$ by δ_1. In case III, $d_{ss'}^{c_1 c_2}$ is incremented by δ_2. In case I and case IV, $d_{ss'}^{c_1 c_2}$ remains unchanged.

In case II, the two routes need to be modified by the reinsertion move to convert one solution to the other, while in case III only one route must be changed. Therefore, we give higher weight to the distance in case I then in case II. In the implementation, $\delta_1 = 10$ and $\delta_2 = 1$. If it is desirable to always select the best solution in SS_1, call it s^*, as one of the elements for SS_2, we can always set $\mu_{s*} = 1$ in model (3.3).

Phase II

Each solution in SS_2 is improved with an ALNS that takes an iterative approach using a ruin and recreate paradigm. At each iteration, given an incumbent solution S, μ requests are removed from their routes and placed in the set $U(S)$, where μ is a parameter. The randomized construction procedure outlined in Figure 3.3 is then use to reinsert the elements of $U(S)$ into routes. If the resulting solution has a lower cost than the incumbent, then the incumbent is replace with the new solution. The process is repeated until a predefined number of iterations N_{II}^{max} or time limit T_{II}^{max} is reached. To remove requests from a solution, ALNS randomly selects a removal heuristic rh from the set $\{rh_1, \ldots, rh_{nr}\}$ and applies it μ times. The details are presented in the next section.

The solution provided by ALNS is further improved with RILS. Given an incumbent, RILS exams the reinsertion neighborhood of each client c until a better solution is found or it is determined that none exists. If the first case, the incumbent is updated and the process is repeated; otherwise, it stops. To explore the reinsertion neighborhood of a solution, RILS loops through all requests, removing each one from its current position in turn trying to reinsert it elsewhere in same route or into all other positions in the other routes. Unlike ALNS, RILS searches the neighborhood exhaustively so it better serves as an intensification mechanism. Because it is time consuming, though, we only apply it to the final ALNS solution. The pseudocodes for ALNS and RILS are given in Figures 3.4 and 3.5, respectively.

Procedure `ALNS`

Input: Phase I solution S_I, time out period T_{II}^{max}, max number of iterations N_{II}^{max}, number of requests to be removed μ, removal heuristics $\{rh_1, \ldots, rh_{nr}\}$, insertion heuristics $\{ih_1, \ldots, ih_{ni}\}$.

Output: Best solution found S.

Step 0: Initialization

Set $S = S_I$, run time $T_{run} = 0$, iteration count $iter = 0$;

Set number of requests to be removed at each iteration to μ;

Set initial values of removal heuristics weights $\{rw_1, \ldots, rw_{nr}\}$;

Set initial value of insertion heuristics weights $\{iw_1, \ldots, iw_{ni}\}$;

Step 1: Set $S_{tmp} = S$;

Step 2: Randomly select a removal heuristic rh_i based on their corresponding weights and remove μ clients from routes of S_{tmp}; place removed clients in $U(S_{tmp})$;

Step 3: Set S_{tmp} = `Randomized_Construction`$(S_{tmp}, \{ih_1, \ldots, ih_{ni}\}, \{iw_1, \ldots, iw_{ni}\})$;

Step 4: Update $\{rw_1, \ldots, rw_{nr}\}$, $\{iw_1, \ldots, iw_{ni}\}$, μ;

Update T_{run} and put $iter \leftarrow iter + 1$;

If ($COST(S_{tmp}) < COST(S)$), then set $S = S_{tmp}$; Go to Step 1;

If ($T_{run} > T_{II}^{max}$ or $iter > N_{II}^{max}$), then

Set S = `Reinsertion_LS`(S);

Return S;

Go to Step 2;

Figure 3.4. Pseudocode for ALNS

Procedure `Reinsertion_LS`

Input: Initial solution S_0

Output: Improved solution S

Step 0: Set $S = S_0$;

Step 1: For each client c in the routes of S

Set $S_{tmp} = S$;

Remove c for its routes and add it to $U(S_{tmp})$;

Find best insertion $\wedge_c$ for c into the routes in S_{tmp};

If ($\wedge_c$ exists), then update S_{tmp} with $\wedge_c$, remove c from $U(S_{tmp})$ to get $S_{tmp} = S'(S_{tmp}, \wedge_c)$;

If ($COST(S_{tmp}) < COST(S)$), then set $S = S_{tmp}$, go to step 1;

Step 3: return S.

Figure 3.5. Pseudocode for RILS

Removal heuristics

Our version of ALNS includes the following three removal heuristics.

Shaw removal heuristic (rh_1) tries to select client requests that are in some sense similar to each other. In so doing, there is a higher likelihood that they can be reinserted into different routes giving an overall cost reduction. If the selected requests are significantly different from each other, it is likely that to retain feasibility, most if not all of them will have to be reinserted into their original positions, yielding no improvement.

The heuristic consists of three operations: (1) a request is randomly removed from the existing routes of solution S and placed in $U(S)$; (2) the requests not in $U(S)$ are sorted using a relatedness measure R_c that is defined presently, and (3) one request is randomly selected from the sorted list using probabilities derived from R_c. This process is repeated until the desired number of requests μ are removed.

Shaw (1998) proposed the following relatedness measure for client c currently on a route,

$$R_c = \sum_{c' \in U(S)} \left(\sum_{d \in D} \theta_{1d} \left| L_{cd} - L_{c'd} \right| + \theta_2 \left| D_{i_c i_{c'}} + D_{j_c j_{c'}} \right| + \theta_3 \left(\left| t_{i_c} - t_{i_{c'}} \right| + \left| t_{j_c} - t_{j_{c'}} \right| \right) \right) \tag{3.4}$$

where c' is a client in the set RS, L_{cd} and $L_{c'd}$ are the loads for demand type d for customer c and c', respectively, $D_{i_c i_{c'}}$ and $D_{j_c j_{c'}}$ are the travel distances between the pickup nodes and delivery nodes of two customers, t_{i_c}, $t_{i_{c'}}$, t_{j_c}, and $t_{j_{c'}}$ are route service start times at node i_c, $i_{c'}$, j_c, and $j_{c'}$, respectively, in the current solution, and θ_{1d}, $d \in D$, θ_2 and θ_3 are weights for each component in Eq. (4). By definition, the smaller the value of R_c, the more similar the request of client c is to the requests in $U(S)$.

Random removal heuristic (rh_2) randomly selects clients in routes to remove. Randomness introduces a degree of diversity that helps surmount local optima.

Route random sweep heuristic (rh_3) randomly picks a route and removes all clients on it. This process is repeated until any additional removals result in more clients being removed than specified by the current value of μ. In that case, the *Shaw removal heuristic* is used to bring the number up to μ. By removing a complete route at each step, we increase the possibility of finding a new solution that requires fewer vehicles than the incumbent.

After each ALNS iteration, the weights for each insertion and removal heuristic $\{rw_1, \ldots, rw_{nr}\}$ and $\{iw_1, \ldots, iw_{ni}\}$ are adaptively updated based on the quality of the new solution. The number of requests to be removed μ is dynamically selected from the set $\{\mu_1, \ldots, \mu_m\}$ according to the number of repeated solutions in past iterations. For more detail, see Qu and Bard (2012).

Feasibility check and vehicle type assignment

In ALNS, a series of client insertion and removal operations are carried out repeatedly. After a change is made to a route, we need to update service start times of the affected nodes and check the feasibility of the new routes with respect to time window and vehicle capacity. For this purpose, we track the following variables for each node i in a route: t_i, as defined in model (3.1), is the service start time at node i; z_{id} is the capacity required for each type of demand after visiting node i; and FS_i is a set of vehicle types that can satisfy the capacity requirement on the route after visiting node i. Given a new route that visits the sequence of nodes $\{b, i_1, \ldots, i_{k-1}, i_k, i_{k+1}, \ldots, b'\}$, to check its feasibility we first identify the first node i_k in the sequence whose variables need to be updated. Depending on how the new route was created, i_k is either a new node that resulted from an insertion operation, or an original node that immediately followed a removal operation. We first update the variables associated with i_k using information from its

predecessor node i_{k-1}, and then update the variables associated with the successor node i_{k+1} given the previous node information. The process is repeated until last node in the sequence b' is reached.

Given two consecutive nodes i and j in the same route, variables for j are updated accordingly using the information for node i in the following way.

$$t_j = \max(t_i + T_{ij}, a_j) \tag{3.5a}$$

$$z_{jd} = z_{id} + \Delta_{jd} \tag{3.5b}$$

$$FS_j = FS_i \setminus DS_j \tag{3.5c}$$

In Eq. (3.5c), DS_j is a subset of FS_i such that each vehicle type in DS_j is unable to satisfy at least one the capacity requirements z_{jd}, $d \in D$. To make this determination for vehicle of type $f \in F$, we check the requirement against all possible configurations for the vehicle type. As long as one of the configurations is suitable for all demand types, the vehicle type is acceptable.

Example 3.2. Using the data in Figure 3.6, we demonstrate how the variables are updated. Assume that we have two types of vehicles and each type can be configured to provide two types of capacities. Each unit of the first capacity type ($k = 1$) can accommodate one unit of demand of type $d = 1$ and each unit of the second capacity type ($k = 2$) can accommodate one unit of demand of type $d = 2$. The corresponding capacities for each capacity type under different configurations are given in the figure.

In the solution under consideration, there is a route that visits node i followed by node j. At this point we have updated variables in Eqs. (3.5a) – (3.5c) at node i and need to determine the variable values at node j. Now suppose that node i is visited at time $t_i = 120$. The load on the vehicle after serving node i is 8 units for demand type 1 and 2 units for demand type 2, which can be satisfied by both vehicle types, so we have $FS_i = \{1, 2\}$. The travel time T_{ij} from node i to j including the service time at i is 15. Given that the service time window for node j is [140, 150], node j will be served at time $\max\{120 + 15, 140\} = 140$. At node j, the load requirement is $\Delta_{j1} = 1$ unit for demand type 1 and $\Delta_{j2} = 1$ unit for demand type 2, so the total capacity requirement after serving node j is 9 units for demand type 1 and 3 units for demand type 2. The requirement can be satisfied by configuration 2 of vehicle type 1, but not by either of the configurations of vehicle type 2. Therefore, $DS_j = \{2\}$ and $FS_j = \{1\}$. ■

Vehicle type	Configuration 1 capacity		Configuration 2 capacity	
	$k = 1$	$k = 2$	$k = 1$	$k = 2$
1	12	2	10	3
2	12	2	8	3

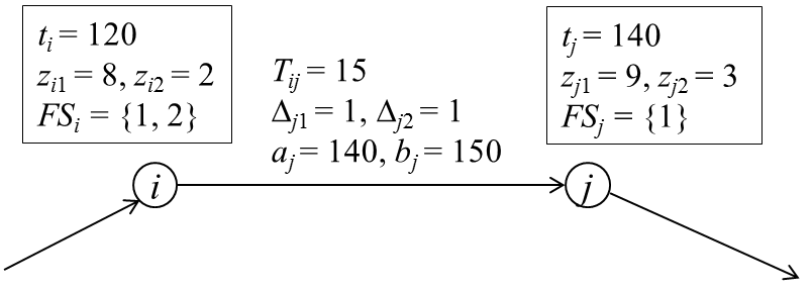

Figure 3.6. Example for updating node information

When one demand type can be satisfied by multiple capacity types, to determine if a given vehicle type f with configuration λ can satisfy the demands at a node j, we need to solve the following generalized assignment problem to determine the amount y_{kd} of each capacity type k used to accommodate demand d for all $k \in K$ and $d \in D$ [in model (3.6), j,z_{jd}, f and λ are fixed inputs].

$$\text{Minimize} \quad \sum_{k \in K} \sum_{d \in D} y_{kd} \tag{3.6a}$$

$$\text{subject to} \quad \sum_{k \in K} q_{kd} y_{kd} \geq z_{jd}, \ \forall\, d \in D \tag{3.6b}$$

$$\sum_{d \in D} y_{kd} \leq Q_{f \lambda k}, \ \forall\, k \in K \tag{3.6c}$$

$$y_{kd} \in Z_{+}^{1}, \ \forall\, k \in K, \ d \in D \tag{3.6d}$$

The objective function (3.6a) minimizes the total capacity assigned. Constraints (3.6b) ensure that each demand type is accommodated and constraints (3.6c) ensure that capacity allocated for each capacity type does not exceed the capacity available for a given vehicle type f and configuration λ. For each vehicle type $f \in FS_i$, if no feasible solution is found for model (3.6) for any of the configurations $\lambda \in \Lambda_f$, we then add f to DS_j.

A route is determined to be infeasible if at any node i, $t_i > b_i$ or $FS_i = \varnothing$. If at node b', $FS_{b'}$ is not empty but does not contain the vehicle type initially assigned to the route, we need to reassign vehicle types to all routes, which we do by solving a transportation problem. If no valid assignment exists, then the new route is invalid. Otherwise, vehicle types are reassigned to routes according to the result. The transportation problem is defined as the follows.

Indices and sets

R set of routes

r index for route

FS_r set of vehicle types that is valid for route r

Parameters

η_f cost associated with vehicle type f

Decision variables

θ_{rf} (binary) 1 when route r is assigned to vehicle type f, 0 otherwise

Model

$$\text{Minimize} \quad \sum_{r \in R} \sum_{f \in FS_r} \eta_f \theta_{rf} \tag{3.7a}$$

$$\text{subject to} \quad \sum_{f \in FS_r} \theta_{rf} = 1, \ \forall\, r \in R \tag{3.7b}$$

$$\sum_{r \in R} \theta_{rf} \le N_f, \ \forall\, f \in F \tag{3.7c}$$

$$\theta_{rf} \in \{0,1\}, \ \forall\, r \in R, \ f \in F \tag{3.7d}$$

The objective function (3.7a) minimizes the cost of the fleet, but in actuality, any feasible solution would be acceptable. In the implementation, we set $\eta_f = 0$ for all f. Constraints (3.7b) ensure that there is a vehicle type assigned to each route and constraints (3.7c) ensure that for each vehicle type, the number of routes assigned is no more than the number available.

Given that models (3.6) and (3.7) are relatively small integer programs, they can be solved in negligible time with any commercial code (of course, (3.7) is polynomially solvable). However, because the feasibility check must be performed repeatedly throughout the MSALNS, the cumulative effort may be excessive. Therefore, we developed a simple swap heuristic to find feasible assignments and only resorted to solving model (3.7) when the heuristic failed. If route r

is currently assigned vehicle type $f_r \notin FS_r$ then it must be assigned a new vehicle type from the set FS_r. The swap heuristic iterates through all other routes in the solution searching for a candidate that can be swapped with f_r. If there exists a route r' with a valid vehicle type set $FS_{r'}$ and is currently assigned vehicle type $f_{r'}$, such that $f_{r'} \in FS_r$ and $f_r \in FS_{r'}$, then the heuristic stops because it has found a valid solution which is realized by reassigning f_r to route r' and $f_{r'}$ to route r.

Similarly, we have developed an efficient heuristic to solve model (3.6). In our problem, there are three demand types: seat, wheelchair and walker, and two vehicle capacity types: seat and wheelchair. The capacity associated with one seat capacity can satisfy the demand for one seat or one walker, while the capacity associated with one wheelchair can satisfy the demand for one wheelchair or two walkers. This problem is trivial to solve. We first determine if there is sufficient capacity for seat and wheelchair demand, and then determine if the remaining capacity can accommodate the walker demand. In a more general setting, one can take a similar approach and develop efficient heuristics for finding solutions to models (3.6) and (3.7), only resorting to solving them optimally when the heuristics fail.

3.5.2 Branch and price and cut

One way to improve client satisfaction is to allow participants to select from specific time slots to be picked up and give a maximum amount of ride time that he or she is willing to tolerate. The HPDPCC problem under this setting has more restricted time windows. Therefore an exact algorithm might be possible. In the following, we propose an exact algorithm based on branch and price and cut for HPDPCC with restricted time windows..

Dantzig-Wolfe Decomposition

Even with more restricted time windows, Model (3.1) is unlikely to be tractable for instances of realistic size. In fact, in the tests we performed using CPLEX, we were only able to solve cases with up to 8 client requests and 2 vans in 2 hours. As an alternative, we use Dantzig-Wolfe (D-W) decomposition to construct a set covering master problem whose columns are generated by iteratively solving a series of subproblems, one for each vehicle type *f*. The subproblems themselves are MIPs, and are solved with a labeling algorithm that makes use of a new dominance rule. To improve the lower bounds provided by the master problem solution, we

dynamically add subset-row inequalities to it after each D-W iteration (Jepsen et al. 2008). The full B&P&C algorithm is described in Section "branch and price".

Master problem

The HPDPCC of interest can be formulated as a set covering problem where variables are routes that serve client requests. Given model (3.1), it is always possible to obtain such a formulation by applying D-W decomposition (e.g., see Wolsey 1998). Below, we use a direct approach and define the set covering master problem with the help of the following additional notation.

Indices and sets

R_f set of routes for vehicle type $f \in F$

r index for routes; $r \in R_f$

Parameters

c_{fr} cost of route $r \in R$

c_c cost of unsatisfied client request $c \in C$

a_{rc} 1 when route r serves client c, 0 otherwise

Decision variables

v_r (binary) 1 when route r is selected, 0 otherwise

o_c (binary) 1 when client c is not served, 0 otherwise

Model

$$\text{Minimize} \quad \sum_{f \in F} \sum_{r \in R_f} c_{fr} v_r + \sum_{c \in C} c_c o_c \tag{3.8a}$$

$$\text{subject to} \quad \sum_{r \in R_f} v_r \le n_f, \ \forall f \in F \tag{3.8b}$$

$$\sum_{f \in F} \sum_{r \in R_f} a_{rc} v_r + o_c = 1, \ \forall c \in C \tag{3.8c}$$

$$v_r \in \{0,1\}, \ \forall r \in R_f, \ f \in F; \ o_c \ge 0, \forall c \in C \tag{3.8d}$$

The objective function (3.8a) is aimed at minimizing the total cost of the routes plus the cost of not serving a subset of clients. The route coefficient c_r includes the cost associated with driving time, passenger ride time and vehicle cost. Unlike model (3.1), we have now assigned a high cost c_c for each client request $c \in C$ that is not served. This alleviates the need to find initial

routes that visit all clients when column generation is started; it is equivalent to the "Big-*M*" formulation used to construct an initial basis in linear programming.

Constraints (3.8b) ensure that the number of routes selected for a given vehicle type does not exceed the number of available vehicles of that type, and are equivalent to (3.1b). Constraints (3.8c) guarantee that all client requests are either served or left open, and are the counterpart to (3.1c) – (3.1d).

To avoid the need to explicitly generate all feasible routes *F*, model (3.8) is typically solved by column generation, which starts with an initial set of feasible columns, or routes in our case. Then one or more routes are generated by solving a pricing problem to find columns that have the potential to improve the objective function when added to model (3.8). This process is repeated until no additional columns can be found that show promise.

It is worth mentioning that when solving the relaxation of the model (3.8), we don't explicitly impose the upper bound $v_r \leq 1$; instead we only require $v_r \geq 0$ because the upper bound is implied by constraints (3.8c). Otherwise, we would have to consider the dual variables associated with $v_r \leq 1$ in the pricing problem.

For real instances, model (3.8) cannot be solved directly because it is impractical to enumerate all feasible columns or routes. Instead, we start with a subset of columns and solve the corresponding *restricted master problem* as an LP rather than as an IP. The optimal dual variables are used to construct a function that represents the reduced cost of a generic column in the master problem. Minimizing this function over all feasible routes allows us to identify columns not in the restricted master problem that have negative reduced cost. Such columns are added to model (3.8), which is then re-solved. The procedure is usually embedded in branch and bound (giving rise to branch and price) to get optimal integer solutions.

Pricing problem

To find promising columns to add to the restricted version of model (3.8), we solve one subproblem for each vehicle type *f*. That is, for each vehicle type *f* we try to find a route *r* that has a negative reduced cost given by $\bar{c}_{fr} = c_{fr} - \mu_f - \sum_{c \in C} a_{rc}\pi_c$, where μ_f, π_c are the dual variables associated with constraints (3.8b) and (3.8c), respectively. The route cost c_{fr} expressed in terms of original problem variables is

$c_{fr} = \sum_{(i,j)\in A} \sum_{\lambda\in\Lambda_f} c_{fij} x_{ij}^{f\lambda} + \sum_{c\in C} \tau_c (t_{j_c} - t_{i_c}) \sum_{(i_c,j)\in A} \sum_{\lambda\in\Lambda_f} x_{i_c j}^{f\lambda}$ and $a_{rc} = \sum_{(i_c,j)\in A} \sum_{\lambda\in\Lambda_f} x_{i_c j}^{f\lambda}$.

Therefore the reduced cost in terms of original problem variables is the following.

$$\bar{c}_{fr} = \sum_{(i,j)\in A} \sum_{\lambda\in\Lambda_f} c_{fij} x_{ij}^{f\lambda} + \sum_{c\in C} \tau_c (t_{j_c} - t_{i_c}) \sum_{(i_c,j)\in A} \sum_{\lambda\in\Lambda_f} x_{i_c j}^{f\lambda} - \mu_f - \sum_{(i_c,j)\in A} \sum_{\lambda\in\Lambda_f} \pi_c x_{i_c j}^{f\lambda} \tag{3.9}$$

The pricing problem is defined below, where most constraints have counterparts in model (3.1).

Subproblem f

$$\text{Minimize} \sum_{(i,j)\in A} \tilde{c}_{fij} \sum_{\lambda\in\Lambda_f} x_{ij}^{f\lambda} + \sum_{c\in C} \tau_c \left(t_{j_c} - t_{i_c}\right) - \mu_f \tag{3.10a}$$

subject to

Vehicle flow balance

$$\sum_{\lambda\in\Lambda_f} \sum_{j\in NF(b)} x_{bj}^{f\lambda} = 1 \tag{3.10b}$$

$$\sum_{j\in NB(i)} x_{ji}^{f\lambda} - \sum_{j\in NF(i)} x_{ij}^{f\lambda} = 0,\ \forall\ i \in N \setminus \{b, b'\},\ \lambda \in \Lambda_f \tag{3.10c}$$

$$\sum_{\lambda\in\Lambda_f} \sum_{i\in NB(b')} x_{ib'}^{f\lambda} = 1 \tag{3.10d}$$

Client flow balance

$$\sum_{\lambda\in\Lambda_f} \sum_{j\in NF(i_c)} x_{i_c j}^{f\lambda} = \sum_{\lambda\in\Lambda_f} \sum_{i\in NB(j_c)} x_{ij_c}^{f\lambda},\ \forall\ c \in C \tag{3.10e}$$

$$\sum_{j\in NF(i)} z_{ij}^{d} - \sum_{j\in NB(i)} z_{ji}^{d} = \Delta_{id},\ \forall\ i \in N,\ d \in D \tag{3.10f}$$

Vehicle capacity configuration

$$z_{ij}^{d} \le \sum_{k\in K} q_{kd} y_{ij}^{dk},\ \forall\ d \in D,\ (i,j) \in A \tag{3.10g}$$

$$\sum_{d\in D} y_{ij}^{dk} \le \sum_{\lambda\in\Lambda_f} x_{ij}^{f\lambda} Q_{f\lambda k},\ \forall\ k \in K,\ (i,j) \in A \tag{3.10h}$$

Time continuity

$$t_i + T_{ij} \le t_j + \left(1 - x_{ij}\right) T^{max},\ \forall\ (i,j) \in A \tag{3.10i}$$

$$a_i \le t_i \le b_i,\ \forall\ i \in N \tag{3.10j}$$

Variable definitions

$$x_{ij}^{f\lambda} \in \{0,1\},\ \forall (i,j) \in A,\ \lambda \in \Lambda_f;\ y_{ij}^{dk} \in Z_+^1,\ \forall\ d \in D,\ k \in K,\ (i,j) \in A$$

$$z_{ij}^{d} \geq 0,\ \forall\ d \in D,\ (i,j) \in A;\ t_i \geq 0,\ \forall\ i \in N \qquad (3.10k)$$

The objective function (3.10a) is equivalent to (3.9). It represents the reduced cost of a route in (3.8) and consists of three terms. The first term combines the first and last term in (3.9) and represents the cost of traversing each arc (i, j), where $\tilde{c}_{fij} = c_{fij} - \pi_c$ if $i = i_c$, or $\tilde{c}_{fij} = c_{fij}$ otherwise. The second term is equivalent to the second term in (3.9), it's the actual cost associated with passenger travel time. Because model (3.10) is a minimization problem, the term $\sum_{(i_c,j)\in A}\sum_{\lambda\in\Lambda_f} x_{i_c j}^{f\lambda}$ can be dropped from the second term without changing the objective function cost. The last term is an adjustment associated with constraint (3.8b). Constraints (3.10b) and (3.10d) ensure that the route has to start and end at depot, while constraints (3.10c) ensure that vehicle flow balance is maintained at each client node. Flow balanced for each request is guaranteed by constraints (3.10e) and (3.10f). The former ensure that if a client is picked up, he will also be dropped off; the latter ensure that client flow is balanced on the route at each node. If the vehicle visits a pickup node i_c for client c, it will pick up the loads in the set $\{L_{cd} : d \in D\}$; if the vehicle visits a delivery node j_c for client c, then it will drop off the loads in the set $\{L_{cd} : d \in D\}$; otherwise, the load on the vehicle for each demand type remains unchanged.

The capacity restriction for each demand type $d \in D$ is enforced by constraints (3.10g) and (3.10h). In particular, (3.10g) ensure that the quantity of flow on arc (i,j) associated with demand type d must be less than or equal to the total capacity available for the demand provided by all capacity types assigned to the demand type on the arc (i,j); (3.10h) ensure that each capacity type k assigned to arc (i,j) is no more than the amount permitted by the selected configuration of the vehicle type serving that arc.

Constraints (3.10i) enforce the requirement that time is increasing between any two nodes connected by an arc. Time continuity has to be observed for arc (i,j) when the vehicle visits nodes i and j in succession; otherwise, the constraints are nonbinding. An added benefit provided by (3.10i) is that it prevents subtours from arising that don't start and end at the depot. Constraints (3.10j) ensure that the time service begins at a node satisfies the time window restrictions. Finally, variable definitions are given in (3.10k).

Subproblem (3.10) is still NP hard, and again proved difficult to solve directly using CPLEX. We now present and alternative approach.

Labeling algorithm

Each subproblem can be viewed as a resource constrained elementary shortest path problem, which can be solved using a dynamic programming technique called a labeling algorithm. Given a graph $G = (N, A)$ with a source node and a sink node, labeling algorithms build partial paths in the graph, starting at the source node and ending at a node $i \in N$. The existing partial paths are extended along the arcs leaving their end nodes by applying an extension function. A full path or route is found when the end node is the sink node. In principle all possible paths are enumerated by the algorithm; however, some partial paths may be dominated by others with the same end node, and hence can be eliminated from the search. The major components of the algorithm are label construction, the extension functions, and dominance conditions. In our subproblem, the source node is the depot start node b while the sink node is the depot end node b'.

Label definition

A feasible partial path from the depot node b to any node $i \in N$ is represented by a label L_i, where $L_i = \{P_i, C_i, T_i, W_i\}$. Here, P_i is the sequence of nodes visited by the partial path, C_i is the reduced cost for the partial path, T_i is the earliest feasible service start time at node i, and W_i is a set of loads at node i after it is served. In particular, $W_i = \{w_{id}, d \in D\}$, where w_{id} is the load required at node i for demand type $d \in D$. The label definition is used for feasibility checking as well as used to extend the path. It cannot be used, however, to provide efficient dominance conditions, which help to reduce the number of paths to be enumerated.

A label L_i^1 is said to be dominated by another label L_i^2 if for any feasible path extended from L_i^1, we can find a feasible path extended from L_i^2 with the same or lower cost. We now revise the label definition in order to develop more efficient dominance conditions. Following the ideas in Dumas et al. (1991), let RC_i be the set of client requests whose pickup nodes are in path P_i but have yet to be dropped off. Given RC_i, W_i can be easily calculated as $w_{id} = \sum_{c \in RC_i} L_{cd}$. We also define an additional resource VL_i as a set of valid vehicle configurations for the partial path. To ensure that we only generate elementary paths in which no nodes are visited more than once, we applied the concept of unreachable nodes introduced in Feillet et al. (2004).

Specifically, let UN_i be the set of nodes that cannot be visited by any path extended from the current partial path P_i including those nodes that have been served in the current path. This leads to the new label definition $L_i = \{RC_i, UN_i, C_i, T_i, VL_i\}$, which is integral to determining dominated partial paths.

Dominance condition

To lay the groundwork, we need to define the delivery triangle inequality, which was first introduced by Dumas et al. (1991) and then extended by Ropke and Cordeau (2009).

Definition 3.1: Given an elementary shortest path problem with pickup and delivery requirements defined on a graph $G = (N, A)$, if the associated arc costs satisfy $c_{ij} + c_{jk} \geq c_{ik}$, where $i, j, k \in N$, $(i, j), (j, k) \in A$ and j is a delivery node, then we say that the arc costs satisfy the triangle delivery inequality.

Claim 3.1: If the original arc costs c_{fij}, $(i,j) \in A$ satisfy triangle inequality, then the arc costs $\tilde{c}_{fij}$ in (3.10a) for all $(i,j) \in A$ satisfy the delivery triangle inequality.

Proof. We know that $\tilde{c}_{fij} = c_{fij} - \pi_c$ if $i = i_c$, or $\tilde{c}_{fij} = c_{fij}$ otherwise. Now let α_i equal $-\pi_c$ if $i = i_c$, or 0 otherwise, we have $\tilde{c}_{fij} = c_{fij} + \alpha_i$. Given a delivery node j_c, and arcs (i, j_c), $(j_c, j) \in A$, we have

$$\begin{aligned}
\tilde{c}_{fij_c} + \tilde{c}_{fj_c j} &= (c_{fij_c} + \alpha_i) + (c_{fj_c j} + \alpha_{j_c}) \\
&\geq c_{fij} + \alpha_i + \alpha_{j_c} && (c_{fij} \text{ satisfies triangle inequality}) \\
&= \tilde{c}_{fij} && (\alpha_{j_c} = 0)
\end{aligned}$$

Therefore arc costs $\tilde{c}_{fij}$ satisfy delivery triangle inequality■

Proposition 3.3: Given two labels $L_i^1 = \{RC_i^1, UN_i^1, C_i^1, T_i^1, VL_i^1\}$ and $L_i^2 = \{RC_i^2, UN_i^2, C_i^2, T_i^2, VL_i^2\}$, L_i^1 dominates L_i^2 if $RC_i^1 \subseteq RC_i^2$, $UN_i^1 \subseteq UN_i^2$, $C_i^1 \leq C_i^2$, $T_i^1 \leq T_i^2$ and $VL_i^1 \supseteq VL_i^2$.

Proof. Given $RC_i^1 \subseteq RC_i^2$, we know that $W_i^1 \leq W_i^2$. For any feasible sequence of nodes S_2 extended from L_i^2 that forms a feasible route r_2, we can create a new sequence S_1 by removing the delivery nodes whose associated customers are not in RC_i^1 from S_2. Then S_1 is a feasible

sequence extended from L_i^1 that forms a feasible route r_1 because $UN_i^1 \subseteq UN_i^2, T_i^1 \leq T_i^2$, $W_i^1 \leq W_i^2$ and $VL_i^1 \supseteq VL_i^2$. Now if we have $c_{r_1} \leq c_{r_2}$, then for any feasible route extended from L_i^2 we can find a feasible route extended from L_i^1 with same or better cost, therefore L_i^1 dominates L_i^2.

In the following, we show that the cost incurred in the sequences in terms of objective function (3.10a) satisfy $C_{S_2} \geq C_{S_1}$. Without loss of generosity, we have $S_2 = i, j_{c_1}, ..., j_{c_m}, j, ..., k, b'$ with *m* delivery nodes between nodes *i* and *j*, and sequence $S_1 = i, j, ..., k, b'$ which is obtained by removing the delivery nodes between nodes *i* and *j* from S_2. The clients $c_1, \ldots, c_m$ belong to the set $RC_i^2 \setminus RC_i^1$.

There are three terms in objective function (3.10a). The third term $-\mu_f$ is a constant. It's has been treated as a fixed cost for any label in the algorithm. So given a sequence *S*, the object function cost is $C_S = C_S^I + C_S^{II}$, where C_S^I and C_S^{II} correspond to the first two terms in (3.10a), respectively. From *Claim* 1 we know that $\tilde{c}_{ij}$ satisfies the delivery triangle inequality, so we have $C_{S_1}^I = \tilde{c}_{fij_{c_1}} + ... + \tilde{c}_{fj_{c_m}j} + ... + \tilde{c}_{fkb'} \geq \tilde{c}_{fij} + ... + \tilde{c}_{fkb'} = C_{S_2}^I$.

Now consider the second term. Let φ be the number of delivery nodes after node *j* and including node *j* in both sequences. Let the cost incurred after node *j* in terms of the second term of the objective function in both sequences as ω. Because of triangle inequality and additional deliveries in sequence S_2, we have $C_{S_1}^{II} = \varphi\tau T_{ij} + \omega \leq \varphi\tau(T_{ij_{c_1}} + ... + T_{j_{c_m}j}) + \omega \leq C_{S_2}^{II}$. Consequently $C_{S_2} \geq C_{S_1}$, and when combined with $C_i^1 \leq C_i^2$, we have $c_{r_1} = C_{S_1} + C_i^1 \leq C_{S_2} + C_i^2 = c_{r_2}$, therefore L_i^1 dominates L_i^2 ■

Extension functions

For node *i*, label $L_i = \{RC_i, UN_i, C_i, T_i, VL_i\}$ can be extended along arc (*i*,*j*) if the arc exists and $j \notin UN_i$. The resulting label is $L_j = \{RC_j, UN_j, C_j, T_j, VL_j\}$. Now letting IVL_j be the set of vehicle configurations for vehicle type *f* that cannot accommodate the total demands at node *j*, the extension functions are defined as follows.

$$RC_j = \begin{cases} RC_i \cup \{c\}, \text{ if } j = i_c \\ RC_i \setminus \{c\}, \text{ if } j = j_c \\ RC_i, \text{ otherwise} \end{cases} \tag{3.11a}$$

$$UN_j = UN_i \cup \{j\} \tag{3.11b}$$

$$C_j = C_i + \tilde{c}_{fij} + \sum_{c \in RC_i} \tau_c T_{ij} \tag{3.11c}$$

$$T_j = \max(T_i + T_{ij}, a_j) \tag{3.11d}$$

$$VL_j = VL_i \setminus IVL_j \tag{3.11e}$$

If $T_j > b_j$ (closing time window of node j) or VL_j is empty, then the partial path represented by L_j is infeasible and can be removed. If $j = i_c$ and $T_j + \sigma_j + T_{jj_c} > b_{j_c}$ then there won't be enough time to drop off client c so again the label can be removed. To provide a lower bound on the minimum time expected to accommodate all clients in RC_j, we use $T_{lb} = \sum_{c \in RC_j} \sigma_{j_c} + \max_{c \in RC_j} \{T_{jj_c} - \sigma_{j_c} + T_{j_c b'}\}$ which is the time to service all clients in RC_j plus the longest time required to drop off one of the clients in RC_j and return to the depot. When the current visiting time T_j at node j plus this lower bound for the time to drop off all remaining clients exceeds the depot closing time, that is, $T_j + T_{lb} > b_{b'}$ implying that the label L_j can be removed.

When one demand type can be satisfied by multiple capacity types, to determine if a given vehicle type f with configuration λ can satisfy the demands at node j, we need to solve the generalized assignment problem as defined in model (3.6). For each vehicle configuration $\lambda \in \Lambda_f$, if no feasible solution is found to model (3.6), we add λ to IVL_j. In our problem, there are three demand types: seat, wheelchair and walker, and two vehicle capacity types: seat and wheelchair. The capacity associated with one seat can satisfy the demand for one seat or one walker, while the capacity associated with one wheelchair can satisfy the demand for one wheelchair or two walkers. This problem is trivial to solve. We first determine if there are sufficient capacities for seat and wheelchair demand, and then determine if the remaining capacity can accommodate the walker demand. We use an example to illustrate the use of model (3.6).

Example 3.3: Assume that there are two vehicle configurations for vehicle type f: configuration λ_1 has capacity for 8 seats and 2 wheelchairs; configuration λ_2 has capacity for 4 seats and 4 wheelchairs. We wish to extend label L_i to L_j along arc (i, j). Assume that the demand associated with L_i is 4 seats and 2 wheelchairs so the set of valid configurations is $VL_i = \{\lambda_1, \lambda_2\}$. Further assume that the demand at node j is 1 seat. Accordingly, we need to determine if both configurations are still suitable for the given demands. Solving model (3.6), or by inspection in this case that we see that λ_2 cannot accommodate a 5-seat demand, which implies that $IVL_j = \{\lambda_2\}$ and $VL_j = \{\lambda_1\}$. ■

Figure 3.7 contains the pseudocode for the labeling algorithm. As mentioned, the label definition, the dominance conditions, and the extension functions are the essential components of the algorithm. Important issues for the implementation are the order in which the labels are explored and when the dominance conditions are applied. In our case, we use a priority queue structure PQ to govern the order in which labels are extended. The label with the highest priority is at the top of the queue and therefore is extended first. One successful approach for specifying the priority (Irnich and Desaulniers 2005) is to find a resource that is nondecreasing along the path. By giving the label with the least amount of the resource the highest priority, we can ensure that a label examined earlier in the process will never be dominated by labels explored later. This ensures the most efficient use of dominance conditions.

The resource used for this purpose in our implementation is the node visiting time T. When two labels have the same T values, we give higher priority to the one with the smaller reduced cost, and if they are tied, we select the one with the fewer number of nodes visited.

One difficulty with this scheme is that we are exploring the graph in a manner similar to the breath-first search, so a large amount of computer memory is required to store all the unexplored labels. As a consequence, even with current technology, we are very likely to run out of memory when solving real instances. To resolve this issue, we define the priority of a label in follows. Higher priority is given to labels with smaller T if $T \le T_{bd}$, where T_{bd} is a predefined parameter (60 in our implementation). Otherwise, the priority is determined by the length of the path, i.e., the number of nodes it has visited. The more nodes that are visited along a path, the higher the priority. Also labels with $T \le T_{bd}$ are assigned higher priority than ones with $T > T_{bd}$. For example, suppose that we have four labels L^1, L^2, L^3 and L^4 with the following visit times T

and number of nodes visited NL: $T^1 = 70$, $NL^1 = 2$; $T^2 = 20$, $NL^2 = 5$; $T^3 = 80$, $NL^3 = 6$; $T^4 = 15$, $NL^4 = 3$. If T_{bd} is set to 60, then the label order from highest priority to the lowest is L^4, L^2, L^3 and L^1.

The reasoning behind this ordering approach is that when a label has a small T value, the possible number of route extensions is large so the dominance conditions may be very effective in eliminating many successive paths. When a label has a large T value there are fewer possibilities for extending the route at the current node so the dominance conditions are less likely to provide a large reduction in the state space that remains to be explored. Furthermore, the benefit of the dominance conditions may be outweighed by the time it consumes. Under our proposed priority rule, at the beginning of the labeling algorithm, the labels with $T \leq T_{bd}$ are explored similarly as in a breadth-first traversal when dominance condition are most effective. At later stage, when all labels to be explored have T more than T_{bd}, the algorithm traverse as a depth-first traversal and has least requirement on memory.

To check the dominance conditions, we use a node-label association data structure LS to store the nondominated labels LS_i at each node i. For large instances, however, it is impossible to maintain all such labels so we only store up to a fixed number (250 in the implementation) of labels – those with the smallest T values. Of course, it is still necessary to carry forward all components of a label because each is used in the dominance check. Once a label is created, it is put on the priority queue to be extended later. It may also be stored in the node-label association data structure. We use a smart pointer structure (Alexandrescu 2001) to eliminate the need to store two copies of the same label in both PQ and LS, instead that we only keep references to the label in PQ and LS. A smart pointer keeps one copy of the actual label in memory and maintains a reference count for the number of instances where the actual label is being referred. When a reference to the label is removed from PQ or LS, the reference count decrements by one. When the reference count is reduced to zero the label is removed from the memory.

Procedure `Solve_Subproblem`

Input Graph $G = (N, A)$ with source node b and sink node b'

Output Best path found L^*

Step 0: Initialization

Let $RC^0 = \varnothing$, $UN^0 = \{b\}$, $C^0 = -\mu_f$, $T^0 = a_b$, VL^0 contains all vehicle configurations;

Let $L^0 = \{RC^0, UN^0, C^0, T^0, VL^0\}$;

Let $L^* = L^0$;

Initialize priority queue by putting $PQ \leftarrow L^0$;

Initialize node label association structure $\{LS_i, i \in N\backslash\{b'\}\}$, where LS_i is the set of labels associated with node i and initially are all empty;

Step 1: If $(PQ = \varnothing)$

Remove label L from top of the queue PQ, set i as the last node visited by L;

For each arc (i,j) extended from node i,

Get label $L_j = \{RC_j, UN_j, C_j, T_j, VL_j\}$by applying the extension function to L along (i,j);

If L_j is feasible

Check if L_j is dominated by labels in LS_j;

If L_j is not dominated, then

put $PQ \leftarrow PQ \cup \{L_j\}$;

put $LS_j \leftarrow LS_j \cup \{L_j\}$;

If $(j = b'$ and $C_j < C^*)$, then set $L^* = L_j$;

Go to Step 1.

Step 2: Return L^*;

Stop.

Figure 3.7. Pseudocode for labeling algorithm

Valid inequalities

When column generation converges, we add valid inequalities to the master problem to improve the lower bound obtained. At least two types of cuts have been developed for vehicle routing problems (VRPs) for this purpose. The first are formed by taking linear combinations of the

subproblem variables x_{ij}. Ropke and Cordeau (2009) applied several of these to the PDP, including 2-path inequalities (Kohl et al. 1999), rounded capacity inequalities, precedence inequality, and fork inequalities. Within the framework of D-W decomposition, such inequalities only affect the arc costs in the subproblem and so are easy to accommodate. The second type of cuts are defined on the master problem variables. Usually, one needs to take special care when solving the subproblems to properly account for the dual variables associated with these cuts and to avoid regenerating existing columns. For VRPs, Jepsen et al. (2008) proposed subset-row (SR) inequalities, which can be derived from Chvatal-Gomory (CG) inequalities (Nemhauser and Wolsey 1988), while Spoorendork and Desaulniers (2010) and Bard et al. (2012) used clique inequalities, all defined on master problem variables. In this work, we limited our cuts to subset-row inequalities.

Given a scalar $u_c \geq 0$ for all $c \in C$, by applying the Chvatal-Gomory rounding procedure to (3.8c), we get

$$\sum_{f \in F} \sum_{r \in R_f} \left\lfloor \sum_{c \in C} u_c a_{rc} \right\rfloor v_r \leq \left\lfloor \sum_{c \in C} u_c \right\rfloor \tag{3.12}$$

To specialize Eq. (3.12) to our problem, let $CS = \{c_{1, \ldots,} c_{|CS|}\}$ be a subset of clients, set $u_{c_1} = \ldots = u_{c_{|CS|}} = 1/k$, and for $c \in C \setminus CS$, set $u_c = 0$. This leads to

$$\sum_{f \in F} \sum_{r \in R_f} \left\lfloor \sum_{c \in CS} a_{rc} / k \right\rfloor v_r \leq \lfloor |CS| / k \rfloor \tag{3.13}$$

Jepsen et al. (2008) suggested that the most effective SR inequalities are those with $k = 2$ and $|S| = 3$ (our testing corroborated this finding). Given a fractional solution to model (3.8), we identify violations of Eq. (3.13) by enumerating all client combinations consisting of cardinality three and check their feasibility. All violated inequalities are added to the master problem. The labeling algorithm is then called to solve model (3.10) in an effort to find routes (columns) with negative reduced costs. To account for the cuts added to the master problem, let *SS* be the set of constraints corresponding to (3.13) and η_s be the associated dual variables for $s \in SS$. CS_s is the set of clients associated with *s*. Given a route *r*, we need to calculate $\left\lfloor \sum_{c \in CS_s} a_{rc} / k \right\rfloor$ to determine the corresponding coefficient in constraint *s*. In terms of the variables in model (3.10),

the coefficient is given by $\left\lfloor \sum_{i_c \in CS_s} \sum_{(i_c, j) \in A} x^{\lambda}_{i_c j} / k \right\rfloor$. The objective in (3.10a) is then replaced with

$$\text{Minimize} \sum_{(i,j) \in A} \tilde{c}_{fij} x^{\lambda}_{ij} + \sum_{c \in C} \tau_c \left(t_{j_c} - t_{i_c} \right) - \mu_f - \sum_{s \in SS} \left\lfloor \sum_{i_c \in CS_s} \sum_{(i_c, j) \in A} x^{\lambda}_{i_c j} / k \right\rfloor \eta_s \qquad (3.10a')$$

The last term is the cost adjustment introduced by the SR inequalities. In the labeling algorithm, we need to account for this cost. Specifically, when a label is extended along an arc, we need consider the corresponding change in cost introduced by this term.

For a label L_i at node i we need to introduce a new resource $UC_{si} = |VC_i \cap CS_s| \; mod \; k$ for each dual variable η_s, where VC_i is the set of clients visited on the partial path up to and including node i. This resource indicates the number of clients that have not been accounted for with regard to the SR inequality s. As can be seen in (3.10a′), for label L_i the cost associated with η_s is $-\left\lfloor \sum_{i_c \in CS_s} \sum_{(i_c, j) \in A} x^{\lambda}_{i_c j} / k \right\rfloor \eta_s$ or equivalently, $-\lfloor |VC_i \cap CS_s| / k \rfloor \eta_s$. Now if we extend label L_i along arc (i,j), in the case where $j = i_c$ and $c \in CS_s$, we have $VC_j = VC_i \cup \{c\}$. If $UC_{si} + 1 < k$, we know $UC_{sj} = UC_{si}+1$ and $\lfloor |VC_j \cap CS_s| / k \rfloor = \lfloor |VC_i \cap CS_s| / k \rfloor$, so there is no cost change associated with η_s from L_i to L_j. Otherwise, $UC_{si} + 1 = k$, so $UC_{sj} = 0$ and $\lfloor |VC_j \cap CS_s| / k \rfloor = \lfloor |VC_i \cap CS_s| / k \rfloor + 1$, implying that the cost change associated with η_s along arc (i,j) is $-\eta_s$. In other cases, $UC_{sj} = UC_{si}$ so there is no cost change associated with η_s. The updated extension function (3.11c) is given by

$$C_j = C_i + \tilde{c}_{fij} + \sum_{c \in RC_i} \tau_c T_{ij} - \sum_{s \in SS, \, j = i_c, c \in CS_s, UC_{sj} = 0} \eta_s \qquad (3.10c')$$

Also, with the new resource UC_{sj}, we have the following dominance condition.

Proposition 3.4: Given two labels $L^1_i = \{RC^1_i, \; UN^1_i, \; C^1_i, \; T^1_i, \; VL^1_i, \; \{UC^1_{si}, s \in SS\}\}$ and $L^2_i = \{RC^2_i, \; UN^2_i, \; C^2_i, \; T^2_i, \; VL^2_i, \; \{UC^2_{si}, s \in SS\}\}$, L^1_i dominates L^2_i if $RC^1_i \subseteq RC^2_i, UN^1_i \subseteq UN^2_i$, $T^1_i \leq T^2_i, VL^1_i \supseteq VL^2_i$, and $C^1_i - \sum_{s \in SS'} \eta_s \leq C^2_i$, where $SS' = \{s \in SS : UC^1_{si} > UC^2_{si}\}$

Proof. Given $RC_i^1 \subseteq RC_i^2$, we know that $W_i^1 \leq W_i^2$. For any feasible sequence of nodes S_2 extended from L_i^2 that forms a feasible route r_2, we can create a new sequence S_1 by removing the nodes that are not in RC_i^1 from S_2. Then S_1 is a feasible sequence extended from L_i^1 that forms a feasible route r_1 because $UN_i^1 \subseteq UN_i^2$, $T_i^1 \leq T_i^2$, $W_i^1 \leq W_i^2$ and $VL_i^1 \supseteq VL_i^2$. Now we need to show $c_{r_1} \leq c_{r_2}$.

There are four terms in objective function (3.10a′). Given a sequence *S*, the object function cost is $C_S = C_S^I + C_S^{II} + C_S^{III}$, where C_S^I, C_S^{II} and C_S^{III} correspond to the first, second and fourth terms in (3a′), respectively. As in proof for *Proposition* 3.3, we can show that $C_{S_1}^I \geq C_{S_2}^I$ by virtue of the delivery triangle inequality and $C_{S_1}^{II} \geq C_{S_2}^{II}$.

Both S_1 and S_2 visit the same set of additional nodes (clients), so the difference in the additional cost related to dual variable η_s is at most $-\eta_s$ for $s \in SS'$. Therefore, we know $C_{S_1}^{III} - \sum_{s \in SS'}(-\eta_s) \leq C_{S_2}^{III}$. Now, given that $C_i^1 - \sum_{S \in SS'} \eta_S \leq C_i^2$, we have

$$\begin{aligned} c_{r_1} &= C_i^1 + C_{S_1}^I + C_{S_1}^{II} + C_{S_1}^{III} \\ &\leq C_i^1 + C_{S_2}^I + C_{S_2}^{II} + C_{S_2}^{III} + \sum_{s \in SS'}(-\eta_s) \\ &\leq C_i^2 + C_{S_2}^I + C_{S_2}^{II} + C_{S_2}^{III} = c_{r_2} \end{aligned}$$

Thus, L_i^1 dominates L_i^2. ■

Branch and Price

Column generation terminates with a relaxed solution to the master problem given by model (3.8). If the solution is integral, then it is optimal to the original problem given by model (3.1); otherwise, branch and bound is used to find integer solutions. At each node in the search tree column generation is applied to find the relaxed solution. This procedure is called branch and price. The fact that we are adding cuts to the master problem during column generation leads to branch and price and cut. In this section, we discuss the components of our B&P&C algorithm, including branching and ways to improve performance.

Branching rules

After much experimentation, a hierarchical branching scheme was chosen for the implementation. First we branch on the number of vehicles used when the sum of selected routes $\sum_{r \in R_f} v_r$ for vehicle type *f* is fractional. If there are multiple cases, we select the one whose fractional component is closest to 0.5. Ties are broken arbitrarily. If no fractional vehicle count is found, we apply a client assignment-based branching scheme similar to the one proposed by Ryan and Foster (1981). Given a fractional solution to (3.8), we try to identify a pair of clients (c_1, c_2) in routes such that $\sum_{f \in F} \sum_{r \in R_f} a_{rc_1} a_{rc_2} v_r$ is fractional. When $\sum_{r \in R_f} v_r$ is fractional, such a client pair is guaranteed to exist. In those cases in which $\sum_{r \in R_f} v_r$ is integral but two or more v_r are fractional, such solutions can be converted to integer solutions by inspection. For example, say v_{r_1} and v_{r_2} are fractional with corresponding routes r_1 and r_2 but $v_{r_1} + v_{r_2}$ is integral (the sum must be 1 in this case), where $r_1, r_2 \in R_f$. From arguments in Ryan and Foster (1981), we know that the clients on routes r_1 and r_2 must be identical or there would exist a third fractional route $r_3 \in R_f$ which contained a subset of clients that matched some of the unique clients on either r_1 or r_2. Now, given that we have an optimal relaxed solution, we know that the costs associated with routes r_1 and r_2 must be identical so we can pick one of them, say r_1, and set $v_{r_1} = 1$ and $v_{r_2} = 0$; if the costs were not identical and, say $c_{r_1} < c_{r_2}$, then this would contradict the optimality of the relaxed solution because v_{r_1} should have been 1 instead of fractional. In the implementation, we again select the client pair with fractional $\sum_{f \in F} \sum_{r \in R_f} a_{rc_1} a_{rc_2} v_r$ closest to 0.5.

When a branching decision is made, we need to create child nodes in the search tree that extend from the current node. Depending on which rule is used, the master problem and subproblems will change accordingly. When the decision is to branch on vehicle type *f*, call this case (i), we create two child nodes: in one node, the right-hand side of the appropriate constraint in (3.8b) of the master problem is updated to $\left\lfloor \sum_{r \in R_f} v_r \right\rfloor$; in the second child node, a lower bound of $\left\lceil \sum_{r \in R_f} v_r \right\rceil$ is placed on the summation in (3.8b). This is equivalent to updating the

left-hand side of (3.8b) to $\left\lceil \sum_{r \in R_f} v_r \right\rceil$. There is no change to the subproblem when branching on vehicle type.

When the decision is to branch on client pair (c_1, c_2), call this case (ii), two new child nodes are also created but in the first node, the clients c_1 and c_2 are assigned to the same route while in the second node, c_1 and c_2 must be on different routes. Routes (columns) in the parent node that violate the branching decisions are excluded from the child nodes. To account for the branching decisions in the subproblems, the labeling algorithm has to be modified. In particular, we need to specify the groups of clients $GP = \{ g_l : l = 1,\ldots,m\}$ which must be served by the same route, where m is number of existing groups based on previous branching decisions. This information is update on the branch where a client pair (c_1, c_2) is required to be served in the same route. We need to create a group, say $g = \{c_1, c_2\}$, find all existing groups that contain $c_1$or c_2 and merge these groups with g .

To illustrate, assume that the existing groups of customers $GP = \{ g_1, g_2, g_3 \}$, where $g_1 = \{ c_1, c_3, c_4\}$, $g_2 = \{ c_5, c_6\}$ and $g_3 = \{ c_2, c_7\}$. Because g_1 contains c_1 and g_3 contains c_2, we need to merge g_1 and g_3 with g to form a group $g' = \{ c_1, c_2, c_3, c_4, c_7\}$. Also, g_1 and g_3 must be removed from GP and g' added to GP. Now, for each client c, we need to specify the set of other clients E_c that must be excluded from the vehicle serving c. On the branch where c_1 and c_2 are served by different vehicles, we add c_1 to E_{c_2} and c_2 to E_{c_1}.

To handle case (i) in the labeling algorithm, we need to introduce an additional resource to the label, call it PC_i, which holds the clients to be picked up at node i. In the extension function, when we consider the transition from node i to a pick up node $j = i_c$, we need to check whether i_c is in PC_i. If so, then we put $PC_j \leftarrow PC_i \setminus \{c\}$, otherwise, we need to determine whether there is a group $g \in GP$ that contains client c. If g exists, then $PC_j \leftarrow PC_i \cup g \setminus \{ c\}$; otherwise, no action is taken. To handle case (ii), we make use of UN_i when the label is extended from i to j. If j is in UN_i, then it can't be extended; when $j = i_c$, UN_j is updated to include pickup and delivery nodes for clients in E_c. The updated dominance condition is given in the following proposition.

Proposition 3.5: Given two labels $L_i^1 = \{RC_i^1,\ UN_i^1,\ C_i^1,\ T_i^1,\ VL_i^1,\ PC_i^1,\ \{UC_{si}^1, s \in SS\}\}$ and $L_i^2 = \{RC_i^2,\ UN_i^2,\ C_i^2,\ T_i^2,\ VL_i^2,\ PC_i^2,\ \{UC_{si}^2, s \in SS\}\}$, L_i^1 dominates L_i^2 if $RC_i^1 \subseteq RC_i^2$, $UN_i^1 \subseteq UN_i^2$, $T_i^1 \leq T_i^2$, $VL_i^1 \supseteq VL_i^2$, $PC_i^1 \subseteq PC_i^2$ and $C_i^1 - \sum_{s \in SS'} \eta_s \leq C_i^2$ where SS' = {$s : S \in SS$, $UC_{si}^1 > UC_{si}^2$ }

Proof. Similar to proof for *Proposition* 3.4. ■

Performance improvement

One way to improve the overall performance of the labeling algorithm is to reduce the state space. This can be done in part by iteratively reducing the time windows between feasible pairs of nodes (see Dumas et al. 1991). After one time window is shortened, it may be that a previously feasible transition becomes infeasible. We also eliminate the arcs between all pairs of nodes associated with two clients when no feasible route exists between them due to time window restrictions or available capacity.

For our problem, where we are transporting elderly clients with medical issues, the following two requirements need to be enforced: (i) clients who live together, who are being dropped off at the same location, and who have the same pickup/drop off time windows should be routed together, and (ii) clients on a vehicle who have the same destination should be dropped off together (this almost goes without saying but the efficiency of the code can be improved by explicitly recognizing this situation). Not surprisingly, these additional requirements help to speed up the labeling algorithm by reducing the state space. To handle the first requirement, we replace clients who have the same pickup and drop off locations and time windows with a dummy client. The service times and load requirements for the dummy client are the sums of those of the original clients.

The second requirement is handled in the extension function (3.11). To extend a label from node *i* to *j*, if $i = j_c$ and there is a set of other clients *CL* in RC_i who have the same drop off location ($CL \neq \varnothing$), then the drop off service time for client *c* has to satisfy $\sigma_i \leq \min\{\sigma_{j_c} : c \in CL\}$. This ensures that passenger travel cost is minimized. Similarly because of the requirement that the clients who at the same destination should be dropped off together, node *j* has to be the drop off node for one of the clients in *CL*, and $\sigma_j = \min\{\sigma_{j_c} : c \in CL\}$.

In the formulation of model (3.8), we included penalty columns for the o_c variables. This construct eliminated the need for providing an initial set of feasible routes. Nevertheless, there is ample evidence that the performance of column generation usually can be improved by introducing high quality initial columns in the master problem (e.g., see Bard and Nananukul 2010). After preliminary testing, we decided to use the routes in the solution obtained by adaptive large neighborhood heuristic described in previous section as initial columns at the root node but keep the penalty columns for those cases when the previously obtained solutions were not feasible.

One variation of column generation is not to solve the subproblems to optimality at each D-W iteration but to stop when one or more paths are found whose reduced costs are negative. All such paths have the potential to improve the bounds provided by the master problem. In our case, testing showed that column generation converged faster when this approach was followed. Therefore, the labeling algorithm was terminated when a fixed number (1000 in the implementation) of routes with negative reduced cost were found or when a route was uncovered that priced out negatively after a predefined time limit (100 seconds) was reached for any of the subproblems. Although there is a subproblem for each vehicle type, we don't necessary solve each; we stop with the first subproblem that provides one or more routes whose reduced costs are negative. These routes are then added to the master problem. We also evaluate their feasibility with respect to the other vehicle types and if feasible for say vehicle type *f*, they are added to the set R_f. This strategy reduced the total number of calls to the labeling algorithm, and for similar vehicle types, we found that routes feasible for one type of vehicle were often feasible for those that were similar.

The labeling algorithm is an exact method. Even with effective dominance conditions proposed and efficient implementation, it can be time consuming for instances of realistic size. Therefore, developing efficient heuristics to find routes with negative reduced cost and only resorting to the labeling algorithm when the heuristics fail can improve overall performance. One common idea is to develop a heuristic that uses the labeling algorithm but with relaxed dominance conditions, such as only considering the reduced cost when comparing two labels. Another idea is to perform local search on the columns in the current solutions of the master problem as well as the columns found in the previous iteration of the labeling algorithm. Because

we may terminate the labeling algorithm before it reaches optimality, a local search over the current routes may uncover similar routes whose reduced costs are also negative.

With this in mind, we perform a local search on the existing columns in the master problem after each D-W iteration using the following neighborhood definition: (1) insertion of one or two clients into a route, and (2) removal of one client from a route and then the insertion of one or two new clients into the reduced route. All feasible routes with negative reduced cost that are uncovered at this stage are added to the master problem. Although local search takes little time, we found that it was only effective when the number of columns from the previous iteration was 50 or fewer.

3.6 COMPUTATIONAL EXPERIENCE

3.6.1 MSALNS

The MSALNS heuristic was coded in C++ and run on a PC with 2 GB of memory and an Intel core 2 dual processor with two 2.4 GHz cores. After extensive experimentation, the following parameter settings were seen to provide the best overall performance.

- Parameters related to the number of phase I solutions: $N_0 = 10$, $N_1 = 4$, $N_2 = 2$ [Section 3.5.1]
- Parameters to calculate the distance between solutions: $d_1 = 10$, $d_2 = d_3 = 1$ [Section 3.5.1]
- Parameters for ANLS termination criteria: T_{II}^{max} =1800 sec, $N_{II}^{max} = 2500$ [Section 3.5.1]
- Initial weights for removal heuristics: $rw_1 = 45$, $rw_2 = 45$, $rw_3 = 10$ [Section 3.5.1]
- Initial weights for insertion heuristics: $ih_1 = 20$, $ih_2 = 20$, $ih_3 = 20$, $ih_4 = 20$ [Section 3.5.1]
- Weights to determine the relatedness measure, $\theta_{1d} = 0.2$, $d \in D$, $\theta_2 = 0.1$, $\theta_3 = 1.0$ [Section 3.5.1]
- Weights to determine the constrained measure: $\beta_1 = 0.5$, $\beta_2 = 0.2$, $\beta_{3d} = 0.3$, $d \in D$ [Section 3.5.1]
- Parameters to determine the number of clients to remove in each ALNS iteration: $m = 3$, μ_1, μ_2, μ_3 are 10, 15, 20 percent of number of customer requests in the data respectively [Section 3.5.1]

The most important parameters here are N_2, T_{II}^{max} and N_{II}^{max}. As one might expect, the larger these values, the more thoroughly the feasible region is explored, resulting in better solutions. A balance is necessary, though, between the computational effort and amount of

improvement that can be expected. Based on our experience with real data, we saw little benefit for $N_2 > 2$. In each ALNS run, the time required for further improvement after T_{II}^{max} seconds or N_{II}^{max} iterations increases exponentially. For the removal heuristic weights, we used a relatively small value for rh_3 (sweep heuristic) because it usually affects many requests, and hence greatly increases the amount of time required to reconstruct a solution. Looking at the real data, we observed that the time windows were fairly loose so the corresponding neighborhoods were "smooth." This meant that we only needed to reconstruct a small portion of a solution to find improvements. For the same reason, the number of clients to be removed μ was chosen to be small, ranging from 10% to 20% of the number of requests in the data.

Test data

Due to a lack of benchmark data sets for the problem under investigation, we first used benchmark data set from Parragh (2011) which is for a problem that can be viewed as a special case of our problem with only one configuration setting per vehicle type. To evaluate the performance of our heuristic, we compare our solutions in 5 runs with the ones found in Parragh (2011). We also use the data set to evaluate different settings of our heuristic.

We also used the real instances provided by one of the PACE organizations. We compared the solutions from our heuristic to their solutions, which were constructed manually. Input data are organized in an Excel spreadsheet and include in part client pickup and delivery locations given in terms of latitude and longitude. From those values we calculated the distance between all locations and then the travel time by dividing the distance by miles per hour, which was determined from the formula used by Bard and Jarrah (2009).

To reduce the complexity of constructing routes and schedules, the PACE organization uses the following guidelines: (1) the participants are divided into three groups based on their residential locations and preferred time windows. Each group is given a more restricted time window in which the pickups will be made, which generally begin at 7:30 am and last until 12:00 pm. Specifically, the participants in group 1 are served between 7:30 am and 9:00 am; those in group 2 are served between 9:00 am and 10:30 pm; and the remainder in group 3 are served between 10:30 am and noon; (2) separate vehicles are used to transport the participants who are visiting the center and those who have outside appointments; (3) walker capacity is not taken

into account – it is assumed that walkers can be placed in the isles even though this raises a potential safety risk. In the testing, three sets of problem instances were analyzed to evaluate the performance of MSALNS. The first set was provided by the PACE organization and the other two were randomly generated based on the characteristics of the clients in the PACE database.

We performed two runs with the real data. The first followed current practice of adhering to the time windows of the three groups while the second assumed a single time window between 7:30 am and noon for most participants. In all cases, however, specific pickup and drop off requests were honored. Table 3.1 identifies the number of participants in each test instance. The complete set of input and output files are available from the authors. With respect to vehicles, the collaborating PACE organization owns four types of vans. Each type may be configured in different ways to provide different numbers of seats and wheelchair spaces. The specifics are given in Table 3.2.

To further test the methodology, we randomly generate two additional data sets; call them "A" and "B." Our underlying goal was to determine the impact of guidelines (2) and (3) on the quality of the solution. We performed three runs using data set A, the first with group-specific time windows, and the second and third without. In the third run, we also enforced guideline (2). Each test instance had a total of 100 participants and was designed to reflect the same general characteristics as the real instances, including the percentage requiring wheelchairs or walkers, and the percentage with more specific time windows. The percentage with appointments outside the PACE center varied from 5% to 25% in the instances in data set A. More specifically, there are two instances for each of the following appointment percentages: 5%, 10%, 15%, 20% and 25%.

For each instance in data set B, two runs were made: the first included walker requirements and the second did not. Otherwise, the general characteristics of the real instances were preserved with the exception of the percentage of clients requiring walkers, which now ranged from 20% to 60%. Again, two instances were randomly generated but this time for the following percentages: 20%, 30%, 40%, 50% and 60%.

Table 3.1. Client characteristics of real instances

PACE instance	Number of participants			
	Group 1	Group 2	Group 3	Total
11182011AM	36	29	18	83
01122012AM	34	33	25	92
01272012AM	38	36	23	97
01272012PM	29	19	37	85
03052012AM	50	30	17	97
03052012PM	30	28	40	98
04132012AM	23	25	11	59
04132012PM	20	7	34	61
Average	32	26	26	84

Table 3.2. Vehicle characteristics

Vehicle type	No. available	Configuration 1			Configuration 2		
		No. seats	No. wheelchairs	No. clients	No. seats	No. wheelchairs	No. clients
VAN1	1	1	0	1	0	1	1
VAN2	3	8	2	10	--	--	--
VAN3	6	8	2	10	6	3	9
VAN4	1	8	2	10	4	4	8

Results

DARP instances

Table 3.3 provides the solution comparisons between VNS by Parragh 2011 and our MSALNS for the DARP instances. For the purpose of the comparison, the objective for the DARP instances is to minimize the total routing cost. For each method, we show the solution statistics average over 5 random runs. The statistics are average solution objective (Avg. Obj.), best solution objective in 5 runs (Best Obj.), their deviations from the optimal solution (%) where known and the run time in seconds (CPU). For MSALNS, we also show the time taken to get the solution (CPU*).

Overall our solutions are compared favorably to the ones found by VNS. The average deviations from the best known solutions for average objective over 5 runs are 0.18% for MSALNS comparing to 0.30% from VNS. The average deviation from the best known solutions

for best run is 0.04% for MSALNS which is better than 0.11% for VNS. The average solution time is 151.18 seconds for MSALNS comparing to 115.88 seconds for VNS. This time may be further reduced as indicated by the average run time to find the best solutions, which is 76.26 seconds. In two cases (a3-36 and a4-48) with no optimal solution known, MSALSN is able to improve the best known solutions.

Table 3.4 shows the solution comparisons among different MSALNS settings. For each setting, we show the solution statistics average over 5 random runs. The parameters provided in Section 3.6.1 are used for "MSALNS" setting. Under "1 ALNS" setting, we set N_2 = 1 and N_{II}^{max} = 5000 to allow only one phase I starting solution while allowing second ALNS phase to run for twice amount of iterations. "2 ALNS" setting has the same set of parameters as the ones for "MSALNS" setting except that we picked the best N_2 solutions in terms of the objective in phase I, instead of solving the max diversity problem as in "MSALNS". Overall, the solutions under different settings are comparable and better than the ones obtained by VNS. It shows the robustness of the heuristic. Of these settings, "MSALNS" performs the best indicted by both the deviations of best and average objectives from the best known objectives. "MSALNS" works best for the large instances a3-36, a4-40 and a4-48, while "1 ALNS" works slightly better for smaller instances. This is somewhat expected, long run time of local search under "1 ALNS" setting ensure the extensive exploration of the local neighborhood, therefore the better performance for smaller test instances. However, without multiple starting points, ALNS may get trapped at local minima and fail to find good solutions. The diversified initial phase I solutions under "MSALNS" ensure that the larger solution space are covered by the local search, therefore there are better chance to find global local minimum. If it is necessary to impose an overall limit on the computations, one needs to find a balance between the number of phase I solutions to perform ALNS and the time or iteration limit for each ALNS run. Multiple ALNS runs with different starting solutions help to diversify the search while longer ALNS runs ensure a more extensive exploration of local neighborhoods.

Table 3.3. Solution comparison for DARP instances

Test cases	Best known Obj.	VNS					MSALNS					
		Avg. Obj.	Avg. %	Best Obj.	Best %	CPU (sec)	Avg. Obj.	Avg. %	Best Obj.	Best %	CPU* (sec)	CPU (sec)
a2-16	294.25	294.25	0.00	294.25	0.00	68.40	294.25	0.00	294.25	0.00	4.00	37.08
a2-20	355.74	355.74	0.00	355.74	0.00	141.80	355.74	0.00	355.74	0.00	9.11	77.33
a2-24	431.12	431.12	0.00	431.12	0.00	211.00	431.12	0.00	431.12	0.00	9.62	86.75
a3-18	302.17	302.17	0.00	302.17	0.00	47.20	302.17	0.00	302.17	0.00	6.85	30.50
a3-24	344.83	344.99	0.05	344.83	0.00	83.60	345.23	0.12	345.23	0.12	32.16	85.88
a3-30	494.85	495.13	0.06	494.85	0.00	106.80	494.99	0.03	494.85	0.00	82.32	173.04
a3-36	618.15	619.64	0.24	618.58	0.07	170.60	618.24	0.01	**618.15**	0.00	123.03	246.39
a4-16	299.05	299.05	0.00	299.05	0.00	27.00	299.05	0.00	299.05	0.00	11.77	23.63
a4-24	375.02	376.19	0.31	375.07	0.01	51.60	375.26	0.06	375.07	0.01	29.64	77.21
a4-32	486.93	488.64	0.35	486.93	0.00	88.00	487.50	0.12	486.93	0.00	125.28	197.95
a4-40	557.69	563.34	1.01	561.35	0.66	132.20	565.20	1.35	559.34	0.30	204.72	331.25
a4-48	676.36	687.44	1.64	680.43	0.60	262.40	679.77	0.50	**676.36**	0.00	276.63	447.12
Avg.			0.30		0.11	115.88		0.18		0.04	76.26	151.18

Table 3.4. Solution comparison for DARP instances under different settings

Test cases	1 ALNS				MSALNS				2 ALNS			
	Avg. Obj.	Avg. %	Best Obj.	Best %	Avg. Obj.	Avg. %	Best Obj.	Best %	Avg. Obj.	Avg. %	Best Obj.	Best %
a2-16	294.25	0.00	294.25	0.00	294.25	0.00	294.25	0.00	294.25	0.00	294.25	0.00
a2-20	355.74	0.00	355.74	0.00	355.74	0.00	355.74	0.00	355.74	0.00	355.74	0.00
a2-24	431.12	0.00	431.12	0.00	431.12	0.00	431.12	0.00	431.12	0.00	431.12	0.00
a3-18	302.69	0.17	302.17	0.00	302.17	0.00	302.17	0.00	302.17	0.00	302.17	0.00
a3-24	345.15	0.09	344.83	0.00	345.23	0.12	345.23	0.12	345.93	0.32	344.83	0.00
a3-30	494.99	0.03	494.85	0.00	494.99	0.03	494.85	0.00	495.13	0.06	494.85	0.00
a3-36	618.29	0.02	618.15	0.00	618.24	0.01	618.15	0.00	618.29	0.02	618.15	0.00
a4-16	299.05	0.00	299.05	0.00	299.05	0.00	299.05	0.00	299.05	0.00	299.05	0.00
a4-24	375.03	0.00	375.02	0.00	375.26	0.06	375.07	0.01	375.04	0.01	375.02	0.00
a4-32	488.14	0.25	486.93	0.00	487.50	0.12	486.93	0.00	488.32	0.29	487.99	0.22
a4-40	561.36	0.66	559.34	0.30	565.20	1.35	559.34	0.30	564.63	1.24	561.87	0.75
a4-48	683.48	1.05	682.07	0.84	679.77	0.50	676.36	0.00	682.04	0.84	678.32	0.29
Avg.		0.19		0.10		0.18		0.04		0.23		0.10

Real instances

The computational results are highlighted in Table 3.5 for the eight real instances. Each column summarizes the costs for one of three scenarios: (i) manual solution used by the PACE organization; (ii) MSALNS solution with group-specific time windows; and (iii) MSALNS solution without group-specific time windows. The four columns associated with each scenario report the following costs.

C_t = travel time costs

C_p = passenger ride time costs

C_v = vehicle ownership costs

C_{total} = total cost

Table 3.5 summarizes the percentage difference by cost component between the two MSALNS solutions and the manual solution for each instance. In all cases, MSALNS proved superior. With group-specific time windows the overall improvement averaged 29.61%, and without them, 40.48%.

Table 3.5. Solution summary for real data

PACE instance	Scenario (i) Manual				Scenario (ii) MSALNS				Scenario (iii) MSALNS without groups			
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}
11182011AM	35.33	0.21	0.06	35.60	24.11	0.20	0.04	24.35	21.96	0.19	0.03	22.17
01122012AM	34.40	0.17	0.06	34.63	23.77	0.18	0.04	23.99	18.58	0.21	0.03	18.81
01272012AM	29.59	0.18	0.06	29.82	22.75	0.18	0.04	22.97	18.87	0.19	0.03	19.08
01272012PM	30.24	0.13	0.06	30.42	20.57	0.14	0.03	20.75	18.05	0.11	0.03	18.19
03052012AM	36.39	0.22	0.07	36.68	24.80	0.22	0.05	25.07	20.02	0.22	0.04	20.28
03052012PM	36.82	0.19	0.07	37.08	24.12	0.19	0.06	24.36	20.80	0.18	0.05	21.04
04132012AM	22.27	0.07	0.08	22.42	16.01	0.08	0.03	16.12	13.97	0.10	0.02	14.09
04132012PM	25.01	0.08	0.06	25.15	18.56	0.08	0.04	18.68	15.22	0.07	0.02	15.32
Average	31.26	0.16	0.07	31.48	21.84	0.16	0.04	22.04	18.43	0.16	0.03	18.62

Table 3.6. Comparisons with manual solutions*

PACE instance	Scenario (ii) MSALNS				Scenario (iii) MSALNS without groups			
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}
11182011AM	31.76	4.76	33.33	31.60	37.84	9.52	50.00	37.72
01122012AM	30.90	-5.88	33.33	30.72	45.99	-23.53	50.00	45.68
01272012AM	23.12	0.00	33.33	22.97	36.23	-5.56	50.00	36.02
01272012PM	31.98	-7.69	50.00	31.79	40.31	15.38	50.00	40.20
03052012AM	31.85	0.00	28.57	31.65	44.98	0.00	42.86	44.71
03052012PM	34.49	0.00	14.29	34.30	43.51	5.26	28.57	43.26
04132012AM	28.11	-14.29	62.50	28.10	37.27	-42.86	75.00	37.15
04132012PM	25.79	0.00	33.33	25.73	39.14	12.50	66.67	39.09
Average	29.75	-2.89	36.09	29.61	40.66	-3.66	51.64	40.48

*100% × (Manual – MSALNS) / Manual

Table 3.7 summarizes the computational aspects of MSALNS. For each instance, we present the following measures: the cost of the best phase I solution (columns 2 and 9), the total CPU time for all phase I iterations (columns 3 and 10), the best solution at the end of phase II after local search––RILS (columns 6 and 13), the CPU time at which the best solution was found (columns 7 and 14), and the total CPU time (columns 8 and 15). Additional output includes the cost of the phase I solution for which the best solution was found (columns 4 and 11), and the cost of the solution after ALNS improvement (columns 5 and 12). Based on these statistics, we can make the following observations. Phase II provided significant improvement over phase I but required considerably more time. In 10 out of 16 instances, the best phase I solution produced the best final solution; however, in the other six instances it did not. In 14 out of 16 instances, RILS further improved the ALNS solution, though the improvement was relatively small.

The unit costs used in the analysis were specified by the PACE organization as follows. For drive time c_{time} = \$0.377/min, for passenger ride time τ_c = \$0.0001/min, and for vehicle ownership c_f = \$0.01/van for all f. Evidently, only the drive time cost is real and has highest priority while the other two serve to direct the algorithm to sequentially minimize vehicle use and passenger ride time in that order when multiple optimal solutions exist. One interesting finding is that in almost all cases, the MSALNS solutions have higher passenger ride time costs than the manual solutions. The passenger ride time correlates with participant satisfaction which has lowest priority in the current formulation [see objective function (3.1a)].

With this in mind, we reran the algorithm with a higher unit passenger ride cost to see how this parameter affects the overall cost. Our goal was to determine how much the PACE organization would have to sacrifice in terms of real dollars to increase participant satisfaction. Using a value of 0.01 instead of 0.0001 for τ_c, we get the results reported in Tables 3.7 and 3.8, which are similar to those in Tables 3.3 and 3.4. Again, the MSALNS solutions outperform the manual solutions on all measures. On average, a total savings of 29.02% is realized for scenario (ii) with groups, which is further improved to 35.51% for scenario (iii) when no group classification is imposed. Focusing on the travel time costs alone, C_t, which may be more relevant, we see that there is a price to pay for reducing ride time and hence increasing participant satisfaction. When $\tau_c = 0.0001$, there is a 29.75% average savings with the MSALNS solutions over the manual solutions for scenario (ii) but this value drops to 23.41% when $\tau_c = 0.01$. The difference for scenario (iii) is similar. Using the formula: min = $C_p/\tau_c/$ (no. passengers), the average amount of time a passenger spends on a van for the three scenarios in Table 3.3 is 18.7 min, 19.0 min, 19.1 min, respectively. The corresponding values for Table 3.7 are 18.5 min, 10.9 min, 10.8 min, respectively, a substantial reduction.

Finally, the number of vehicles used for each run can be deduced from the C_v columns in Tables 3.5 and 3.8. For the first four instances, the manual solution required 6 vans, and for the next two instances, 7 vans, and for the last two instances, 8 and 6 vans respectively, independent of the value of τ_c. It is worth mentioning that in all runs, the MSALNS solutions required fewer vans than the manual solution. For scenario (iii), anywhere from 2 to 5 vans were used compared to 6 to 8 in the manual solution. This suggests that the PACE organization can achieve real long-term cost savings by reducing their fleet size by 2 or 3 vans.

Table 3.7. MSALNS performance

	Scenario (ii) MSALNS							Scenario (iii) MSALNS without groups						
			Best solution							Best solution				
Test instance	Best Phase I solution	Phase I time (sec)	Phase I solution	After ALNS	After local search	Best solution time (sec)	Total solution time (sec)	Best Phase I solution	Phase I time	Phase I solution	After ALNS	After local search	Best solution time (sec)	Total solution time (sec)
11182011AM	34.03	50.53	34.03	25.30	24.35	816.28	819.95	33.22	10.05	33.22	22.20	22.17	745.22	1214.08
01122012AM	28.51	17.31	28.51	24.15	23.99	1045.28	1879.23	28.81	193.46	30.19	18.85	18.81	2007.84	2021.86
01272012AM	28.00	163.87	28.00	23.03	22.97	1144.82	1151.31	26.79	9.01	32.16	19.25	19.08	1459.75	1471.15
01272012PM	23.42	33.78	23.42	20.76	20.75	2430.33	4628.83	24.15	79.93	29.41	18.64	18.19	5920.02	6056.30
03052012AM	31.34	159.30	31.34	25.08	25.07	1348.48	1365.20	30.85	14.73	30.85	20.53	20.28	1220.22	2084.35
03052012PM	33.74	26.33	35.19	25.17	24.36	3211.23	3282.98	30.52	457.22	30.52	21.05	21.04	4130.37	4166.91
04132012AM	19.55	3.06	19.55	16.12	16.12	204.27	831.90	18.86	12.43	18.86	14.13	14.09	1742.96	1746.38
04132012PM	20.75	10.49	23.66	18.68	18.68	2702.42	2755.46	19.59	141.00	20.63	15.61	15.32	4750.86	4769.83
Average	27.42	58.08	27.96	22.29	22.04	1612.89	2089.36	26.60	114.73	28.23	18.78	18.62	2747.16	2941.36

Table 3.8. Solution summary with updated passenger ride costs

Test instance	Scenario (i) Manual				Scenario (ii) MSALNS				Scenario (iii) MSALNS without groups			
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}
11182011AM	35.33	18.82	0.06	54.21	29.84	10.26	0.04	40.14	24.73	11.62	0.03	36.38
01122012AM	34.4	17.35	0.06	51.81	24.4	10.83	0.04	35.27	20.81	8.95	0.03	29.79
01272012AM	29.59	17.72	0.06	47.37	24.51	9.67	0.04	34.22	20.28	9.54	0.03	29.85
01272012PM	30.24	12.61	0.06	42.91	22	8.4	0.03	30.43	18.96	8.9	0.03	27.89
03052012AM	36.39	22.09	0.07	58.55	28.38	9.76	0.05	38.19	23.89	9.81	0.03	33.73
03052012PM	36.82	19.14	0.07	56.03	25.45	9.97	0.06	35.48	23.67	10.69	0.05	34.41
04132012AM	22.27	7.22	0.08	29.57	16.94	5.84	0.03	22.81	16.86	5.59	0.02	22.47
04132012PM	25.01	8.46	0.06	33.53	19.66	6.08	0.03	25.77	17.06	5.82	0.02	22.9
Average	31.26	15.43	0.07	46.75	23.90	8.85	0.04	32.79	20.78	8.87	0.03	29.68

Table 3.9. Comparison to manual solutions with update passenger ride costs*

Test instance	Scenario (ii) MSALNS				Scenario (iii) MSALNS without groups			
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}
11182011AM	15.54	45.48	33.33	25.95	30.00	38.26	50.00	32.89
01122012AM	29.07	37.58	33.33	31.92	39.51	48.41	50.00	42.50
01272012AM	17.17	45.43	33.33	27.76	31.46	46.16	50.00	36.99
01272012PM	27.25	33.39	50.00	29.08	37.30	29.42	50.00	35.00
03052012AM	22.01	55.82	28.57	34.77	34.35	55.59	57.14	42.39
03052012PM	30.88	47.91	14.29	36.68	35.71	44.15	28.57	38.59
04132012AM	23.93	19.11	62.50	22.86	24.29	22.58	75.00	24.01
04132012PM	21.39	28.13	50.00	23.14	31.79	31.21	66.67	31.70
Average	23.41	39.11	38.17	29.02	33.05	39.47	53.42	35.51

*100% × (Manual – MSALNS) / Manual

Randomly generated instances

Similar solution summaries for the randomly generated instances are given Tables 3.9, 3.10 and 3.11. Note that the first two digits in the name of each instance indicate the percentage of participants who have an appointment outside the PACE center (data set A) or require walkers (data set B). The results in Table 3.10 are consistent with those obtained for the real instances. In the column labeled "Gap1" in Table 3.10, we report the percentage improvement that was realized when the participants were not divided into groups [scenario (ii) vs. scenario (iii)]. On average, there was a 15.87% reduction in total cost with the [7:30 am, 12:00 pm] time windows.

Additionally, in Table 3.10 we include cost summaries from the MSALNS solutions without group-specific time windows and with guideline (2) enforced [scenario (iv)]; that is, the

participants who have appointments outside the PACE center are picked up and dropped off with different vans than the ones providing transportation directly to and from the center. In the column labeled "Gap2," we report the percentage increase in cost incurred when the participants with outside appointments are served by dedicated vans. On average, there was an 18.01% additional cost for this scenario. By enforcing guideline (2), in all but one case, at least one extra vehicle was required and in most instances, two. Nevertheless, there is no obvious relationship between increased cost and number of outside appointments.

Table 3.11 summarizes the computational aspects of the MSALNS for random generated data set A. As with the real instances, we also found that in 40% of the cases, the best overall solutions were not associated with the best phase I solutions, and that the improvement obtained from RILS was relatively small.

In Table 3.12 we compare solutions for scenario (iii) with those obtained for scenario (v) where group-specific time windows are omitted and walker requirements are ignored. The test instances are associated with data set B. The last column in the table labeled "Gap" gives the percentage increase in cost incurred when accommodating walkers. On average, the increase was 8.32%. Because of the additional capacity requirements for walkers, the vehicles had to drop off clients more frequently to free up space. This is reflected in extra drive time cost, C_t, in the solutions, which are realized when walker capacity requirements are considered [scenario (iii) vs. scenario (v)]. In all but two of the ten instances, passenger ride times, C_p, decreased in the solutions for scenario (v).

Table 3.10. Solution summary for randomly generated instances

Test Inst.	Scenario (ii) MSALNS				Scenario (iii) MSALNS without groups				Scenario (iv) MSALNS without groups; separate vehicles for appointments				Gap1[a]	Gap2[b]
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}		
A051	51.18	0.19	0.05	51.42	43.00	0.20	0.04	43.24	48.00	0.17	0.06	48.24	15.91	11.56
A052	52.23	0.17	0.05	52.46	36.86	0.20	0.05	37.11	49.37	0.18	0.07	49.62	29.26	33.71
A101	52.20	0.20	0.06	52.46	43.99	0.22	0.05	44.25	52.39	0.20	0.07	52.65	15.65	18.98
A102	54.52	0.20	0.05	54.77	45.19	0.19	0.05	45.43	53.23	0.18	0.07	53.48	17.05	17.72
A151	64.97	0.18	0.05	65.20	49.82	0.23	0.06	50.11	61.37	0.19	0.06	61.62	23.14	22.97
A152	53.97	0.20	0.06	54.23	49.64	0.21	0.05	49.90	54.44	0.17	0.07	54.67	7.98	9.56
A201	63.33	0.20	0.06	63.59	53.57	0.23	0.06	53.86	64.36	0.20	0.07	64.63	15.30	20.00
A202	58.97	0.21	0.06	59.24	49.79	0.24	0.06	50.08	58.95	0.18	0.07	59.20	15.46	18.21
A251	68.37	0.17	0.08	68.62	62.42	0.29	0.06	62.78	71.03	0.22	0.08	71.33	8.51	13.62
A252	62.87	0.23	0.07	63.17	55.21	0.28	0.06	55.55	65.23	0.21	0.10	65.53	12.06	17.97
Avg.	58.26	0.20	0.06	58.52	48.95	0.23	0.05	49.23	57.84	0.19	0.07	58.10	15.87	18.01

[a] Gap1 = 100% × (C_{total} in scenario (ii) – C_{total} in scenario (iii)) / C_{total} in scenario (ii)
[b] Gap2 = 100% × (C_{total} in scenario (ii) – C_{total} in scenario (iv)) / C_{total} in scenario (ii)

Table 3.11. MSALNS performance for randomly generated instances

Test inst.	Scenario (ii) MSALNS							Scenario (iii) MSALNS without groups						
			Best solution							Best solution				
	Best Phase I solution	Phase I time (sec)	Phase I solution	After ALNS	After local search	Best solution time (sec)	Total solution time (sec)	Best Phase I solution	Phase I time (sec)	Phase I solution	After ALNS	After local search	Best solution time (sec)	Total solution time (sec)
A051	75.66	223.32	75.66	51.42	51.42	2379.6	2390.94	69.79	51.45	75.54	43.41	43.24	3533.43	3579.54
A052	80.14	59.03	82.57	52.46	52.46	1722.84	1770.89	73.2	114.93	73.2	37.11	37.11	3377.78	3470.41
A101	80.55	29.14	80.55	52.63	52.46	1398.88	2780.69	68.71	107.69	68.71	44.26	44.25	2211.83	4124.42
A102	82.8	171.37	82.8	54.95	54.77	2140.25	4021.61	72.37	162.64	76.37	46.46	45.43	4123.69	4197.88
A151	80.92	269.87	89.42	65.2	65.2	2141.25	4135.51	75.75	302.79	75.75	50.13	50.11	2343.61	4489.28
A152	89.47	283.18	89.47	55.54	54.23	3961.5	4042.11	77.42	100.27	84.32	50.04	49.9	2399.27	4620.73
A201	95.08	63.23	95.08	64.11	63.59	2543.99	4491.85	83.29	426.86	89.92	54.32	53.86	5011.52	5029.79
A202	96.63	390.6	96.63	59.85	59.24	2514.88	4541.34	82.32	354.85	85.19	50.08	50.08	4170.15	4212.32
A251	102.14	427.63	104.17	70.51	68.62	2568.02	4695.83	90.46	333.1	94.93	65.39	62.78	2689.37	4563.45
A252	89.12	380.87	89.12	65.23	63.17	4930.55	4982.15	78.93	313	78.93	56.7	55.55	4932.76	4974.24
Avg	87.25	229.82	88.55	59.19	58.52	2630.18	3785.29	77.22	226.76	80.29	49.79	49.23	3479.34	4326.21

Table 3.12. Solution comparison for randomly generated instances with and without accounting for walkers

Test instance	Scenario (iii) MSALNS without groups				Scenario (v) MSALNS without groups; ignore walker requirements				Gap[a]
	C_t	C_p	C_v	C_{total}	C_t	C_p	C_v	C_{total}	
B201	43.99	0.22	0.05	44.25	43.44	0.22	0.05	43.72	1.21
B202	45.19	0.19	0.05	45.43	42.13	0.27	0.05	42.45	7.02
B301	44.84	0.25	0.05	45.15	43.90	0.27	0.05	44.21	2.13
B302	44.42	0.19	0.05	44.66	42.13	0.27	0.05	42.45	5.21
B401	47.65	0.21	0.05	47.91	43.99	0.29	0.05	44.32	8.10
B402	47.90	0.19	0.05	48.14	42.13	0.27	0.05	42.45	13.40
B501	47.87	0.20	0.05	48.12	42.94	0.22	0.05	43.21	11.36
B502	46.50	0.18	0.05	46.73	40.38	0.23	0.04	40.65	14.96
B601	45.83	0.22	0.05	46.10	43.22	0.21	0.05	43.48	6.03
B602	45.72	0.20	0.05	45.97	39.70	0.23	0.05	39.98	14.98
Average	45.99	0.21	0.05	46.25	42.40	0.25	0.05	42.69	8.32

[a] Gap = 100% × (C_{total} in scenario (iii) – C_{total} in scenario (v)) / C_{total} in scenario (v)

3.6.2 B&P&C

To evaluate the performance of our B&P&C algorithm, we used real data provided by a PACE organization in Wichita, Kansas gathered over the past year on several different occasions. In general, their fleet makes six trips per day to pickup and drop off clients. More specifically, they run three morning trips in the intervals 7:30am-9:00am, 9:00am-10:30am and 10:30-12:00pm, and three afternoon trips in the intervals 1:00pm-2:30pm, 2:30pm-4:00pm and 4:00pm-5:30pm. In current practice, the clients can only select one pickup interval in the morning and one in the afternoon.

To enhance their satisfaction, management would like to evaluate a scenario in which more granular time slots are offered. Accordingly, we have divided the first half hour of each interval into thirds giving the clients nine choices for pickup in the morning and nine for return in the afternoon. For example, the pickup options in the first morning interval are 7:30am-7:40am, 7:40am-7:50am and 7:50am-8:00am. We also limit the ride time to at most one hour so anyone picked up between 7:30 am and 7:40 am, is guaranteed to be dropped off no later than 8:30 am.

To further examine the robustness of the B&P&C algorithm, we randomly generated test data sets with 20 to 50 participants. In naming these data sets, we used five alpha numeric to

identify each. The first symbol is always a "T"; the second and third digits indicate the number of participants, and the final two digits represent the instance; e.g., T2001 denotes the first instance of data set T with 20 participants. In the generation process, we maintained the same characteristics of the real instance in terms of pickup and delivery locations, composition of participants who need wheelchairs or walkers, and available time slots. Input data are organized in an Excel spreadsheet and include in part client pickup and delivery locations given in latitude and longitude. From those values, we calculated the distance between all locations and then the travel time by dividing the distance by miles per hour, which was determined from the formula used by Bard and Jarrah (2009).

Table 3.1 specifies the number of participants in each real instance. The complete set of input and output files are available from the authors. With respect to vehicles, the collaborating PACE organization owns four types of vans. Each type may be configured in different ways to provide different numbers of seats and wheelchair spaces. The specifics are given in Table 3.2.

Real instances

The output statistics provided by the B&P&C algorithm are presented in Table 3.13 for the real instances. Under the broad "Objective" heading, the "Heuristic" column gives the objective values obtained with the ALNS heuristic; the corresponding routes are used as the initial columns for the master problem at the root node of branch and bound tree. The "LP root node w/o cuts" column reports the lower bound obtained by column generation at the root node before the SR cuts being applied. The "LP root node" column gives the strengthened lower bound (Z_{LP}) obtained by column generation at root node after SR cuts are applied. The "Best IP" column indicates the best integer solution obtained (Z_{IP}). This is either the optimal solution when the associated instance is solved to optimality (lower bound = upper bound), or the best primal bound. The "Gap" column, reports the gap between Z_{LP} and Z_{IP}, defined as $(Z_{LP} - Z_{IP})\times 100/Z_{IP}$.

Under the broad CPU heading, the "Heuristic" column gives the time required to get the solution with the ALNS heuristic. The runtime required for column generation at the root node is presented in the "LP root node w/o cuts" column. The runtime required for the final LP solution at the root node after SR cuts are applied is given in the "LP root node" column. The time spent to find the best primal bound is reported in the "Best IP" column and the total time spent by the B&P&C algorithm is given in the "Total" column. The accumulated time spent on

solving the restricted master problems and the subproblems are given in the "MP" and "SP" columns, respectively.

As can be seen in the table, we were able to solve 20 out of 24 real instances to optimality in 3 hours, including the largest case with 38 participants. Optimality was confirmed by either the 0% gap or convergence of branch and bound. In general, the morning instances were easier to solve. This may be attributed to the fact that in the morning pick up take place at individual residences which are widely distributed in the service area, while in the afternoon, most pickups are at the activity center. Thus, there are many more van assignment options and feasible routes in the afternoon even while observing the time window requirements.

The vast majority of the computational effort centered on solving the pricing subproblems; only a small fraction of time was spent on solving the master problem. To improve the performance of the algorithm, we tried various heuristics, such as more relaxed dominance conditions, local searches, to speed up the column generation. The local search heuristic worked best for the instances we tested. With it, we were able to solve several more instances to optimality within the given time limit.

The average objective function gap between the initial heuristic solution and the best IP solution is 1.46%. The heuristic is effective especially in large instances where good solution is obtained by the heuristic while only fraction of the time is used as comparing to B&P&C.

Table 3.14 highlights the statistics for the various components of the B&P&C for the real instances. The "No. nodes" column gives the number of nodes evaluated in branch and bound tree. The "No. iteration" column indicates the number of times that subproblems are solved, and the column labeled "No. columns" gives the total number of columns generated prior to termination. The last two columns report the total number of SR cuts generated and the time required to find them, respectively.

The column generation component of the algorithm proved to be quite strong, especially after including the SR inequalities. Of the 20 instances that were solved to optimality, 14 converged at the root node of the search tree within the given time limit while the others required less than a few dozen nodes. The total time taken to generate the SR inequalities was less than a few seconds and in 18 instances, they improved the lower bound provided by the master problem. In 10 of those instances, the optimal solution was found at the root node.

To further evaluate the performance improvement gained by applying SR inequalities, we rerun the in 10 instance where optimal solution found at the root node and SR inequalities were used without using these inequalities. The average percentage reduction in the total CPU time to explore the nodes in the search tree when SR inequalities were considered was 16.78%. The average number of nodes explored in branch and bound tree is 1 when SR inequalities are applied comparing to 13 when no SR cuts are used.

Randomly generated instances

To confirm the robustness of the B&P&C algorithm and to analyze its performance for different size instances, we randomly generated a set of 20 instances to test. Each reflected the general characteristics of the real instances in terms of percentage of clients who require a wheelchair or walker, and percentage of clients who have appointments. The only parameter that was varied was the number of clients.

The same set of output statistics are presented in Tables 3.15 and 3.16. In 18 out of 20 cases, it can be seen from the 0% gap in Table 3.17 that optimality was attained. In 9 instances, the algorithm converged at the root node with the SR inequalities providing improved lower bound in 6 instances. In 18 instances the lower bounds are improved after adding the SR inequalities.

In general terms, the random instances turned out to be easier to solve than the real instances. We attribute this to the fact that in the former instances the geographical locations of the clients were more sparsely distributed thus requiring longer travel times between clients and therefore fewer routing options for the algorithm to explore.

Figure 3.8 depicts the trend in computation effort as the number of clients increases. The curve is based on the average runtime for instances with 20, 30, 40 and 50 clients. As expected the amount of work required to find solutions increases exponentially with the problem size.

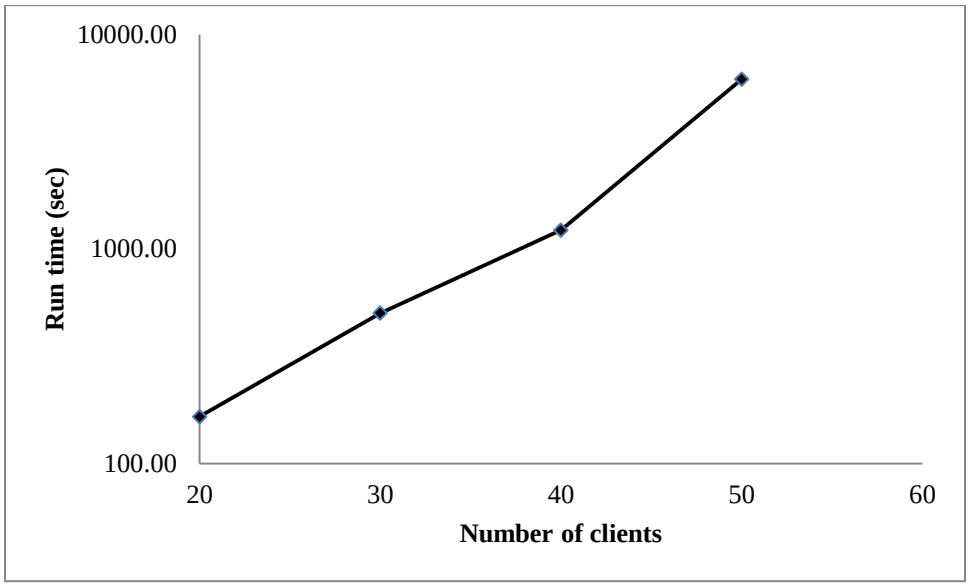

Figure 3.8. Performance trend of the B&P&C algorithm

Instances without configuration options

To investigate the impact of configurable capacity, we ran our algorithm on the same set of real instances without the option to configure the vehicles as best determined. Vehicle configuration 1 in Table 3.2 was used in all cases. Because VAN2, VAN3 and VAN4 have the same capacities, we combined them to form a single vehicle type. This gave us two vehicle types, the first with of one VAN1 and the second with ten VAN2.

Statistics similar to those previously reported are contained in Tables 3.17 and 3.18 for the new runs. Again, 21 out of 24 instances were solved to optimality in 3 hours. In 13 out of the 21 instances, the optimal solutions were found at the rood node. In 18 instances, the SR inequalities helped improve the lower bound obtained at the root node. In 9 instances, optimal solutions were obtained at the root node after adding the SR inequalities. In general, overall performance was comparable to the experiments with configurable vehicle capacity, except for a few difficult instances where the runs now took considerably less time. This can be attributed to the fact that there are now fewer vehicle types to consider, thus fewer calls to the labeling algorithm.

The average percentage cost increase that results when configurable vehicles are not available was 0.31%, so, at least for our data sets, the advantage is minimal. This can be explained by that fact that there is more than sufficient capacity in the fleet to handle the demand; in particular, the portion of clients who need wheelchairs is comparable to the percentage of wheelchair spaces available on the vans. We would expect a larger cost increase if there were fewer vehicles available and more clients who are wheelchair bound. To confirm our hypothesis, we randomly generated 5 test instances where 50% participants are wheelchair bound and 5 test instances with 70% wheelchair bound participants. We also change the objective to give higher weight to the vehicle ownership cost. Intuitively, configurable vehicle capacity could provide more routing options therefore potentially reduce the size of the fleet required. This reduction is of interest to management for long term planning. We performed test runs for each instance with configurable vehicle capacity and with only vehicle configuration 1. The average savings achieved by using configurable vehicle capacity vs. fixed vehicle configuration 1 are presented in Figure 3.9 in terms of the percentage savings of total objective cost and number of vehicles used. The average percentage objective saving by using configurable vehicle capacity increased from 2.35% to 7.84% when percentage of wheelchair bound participants increased from 50% to 70%. Same number of vehicles are used in fixed and configurable vehicle capacity settings for instances with 50% wheelchair bound participants, while 12% less vehicles used under configurable vehicle capacity in cases with 70% wheelchair bound participants. These results align with our expectation. When resource (in this case, wheelchair capacity) is tight under the fixed vehicle configuration but not the other configurations, the advantage of using configurable vehicle capacity is more noticeable.

We also evaluated the impact of available number of configuration options on vehicles on the overall performance of the overall algorithm. We have found that the computation time introduced by additional configuration options is negligible comparing to the overall computation time of the algorithm. The time spent in the heuristic for capacity checks are only a small fraction of the overall computation time.

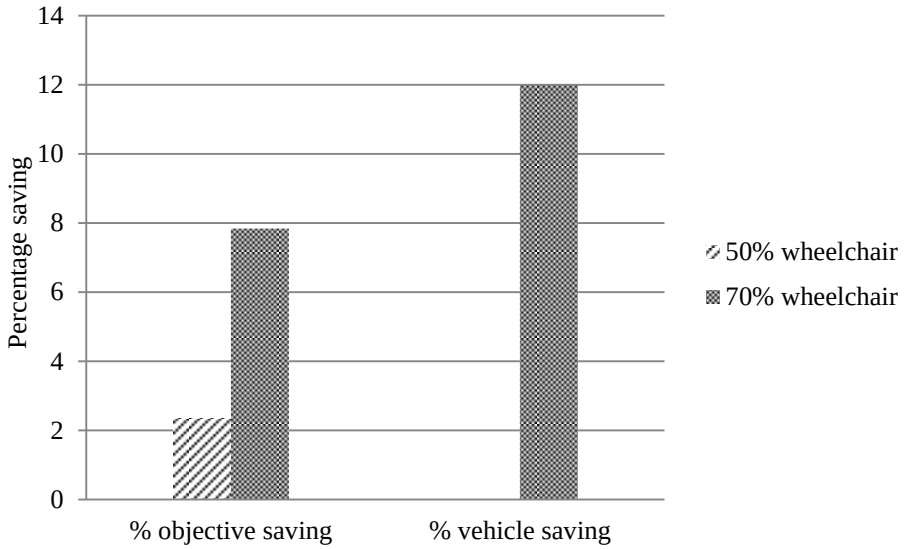

Figure 3.9. Saving trend with configurable vehicle capacity

Table 3.13. Performance of B&P&C algorithm for real instances

	Objective					CPU (sec)						
Instance	Heuristic	LP root node w/o cuts	LP root node	Best IP	Gap (%)	Heuristic	Root node w/o cuts	Root node	Best IP	Total	MP	SP
11182011AMgroup1	14.19	13.35	13.48	13.48	0.00	76.76	658.82	1145.38	1145.38	1145.52	0.93	1051.78
11182011AMgroup2	16.15	16.06	16.11	16.11	0.00	116.70	129.28	133.77	133.77	133.81	0.11	14.91
11182011AMgroup3	3.68	3.63	3.63	3.63	0.00	9.45	9.63	9.63	9.63	9.63	0.01	0.13
01122012AMgroup1	14.25	13.95	13.99	14.00	0.08	151.27	484.86	585.42	1457.72	2430.92	1.83	2222.45
01122012AMgroup2	10.08	9.87	10.01	10.03	0.16	122.39	159.79	181.69	184.53	468.74	1.20	319.94
01122012AMgroup3	7.78	7.65	7.75	7.75	0.00	67.30	79.56	87.07	87.07	87.09	0.08	18.90
01272012AMgroup1	12.12	12.00	12.08	12.09	0.03	112.01	311.76	526.73	536.13	1169.94	0.79	1029.21
01272012AMgroup2	10.60	10.51	10.54	10.54	0.00	144.51	171.85	176.52	176.52	176.58	0.24	26.48
01272012AMgroup3	7.58	7.54	7.56	7.57	0.16	86.94	96.14	99.06	99.10	123.69	0.17	34.04
01272012PMgroup1	9.72	9.62	9.62	9.62	0.00	93.46	5818.44	5818.46	5818.46	5819.01	0.72	5714.80
01272012PMgroup2	3.01	2.97	3.00	3.00	0.00	64.43	113.55	137.86	137.86	137.91	0.08	72.45
01272012PMgroup3	13.57	12.09	12.27	12.30♠	0.19	114.08	3082.78	8732.89	8742.97	20090.75	1.40	19947.65
03052012AMgroup1	17.77	17.40	17.40♣	17.75♠	2.04	194.64	10847.80	10847.80	10880.64	10881.26	5.80	10621.34
03052012AMgroup2	10.93	10.86	10.90	10.90	0.00	98.70	114.73	118.57	118.57	118.59	0.08	18.64
03052012AMgroup3	5.66	5.59	5.64	5.64	0.00	34.78	35.95	36.10	36.10	36.10	0.02	1.18
03052012PMgroup1	11.43	11.03	11.06	11.11♠	0.45	79.16	2454.91	3601.76	3606.19	11502.28	0.80	11409.84
03052012PMgroup2	5.31	5.29	5.29	5.29	0.00	73.12	605.58	605.58	605.58	605.66	0.14	529.63
03052012PMgroup3	13.59	13.04	13.04♣	13.15♠	0.86	126.19	13428.16	13428.16	13436.47	13436.80	0.95	13291.09
04132012AMgroup1	9.91	9.56	9.58	9.68	1.05	147.31	192.20	214.78	216.47	748.01	1.25	571.76
04132012AMgroup2	7.10	7.05	7.07	7.07	0.00	72.77	76.25	76.76	76.76	76.77	0.03	3.60
04132012AMgroup3	3.85	3.80	3.83	3.85	0.49	28.45	29.62	30.79	30.90	34.67	0.05	5.28
04132012PMgroup1	7.67	7.63	7.66	7.66	0.00	112.52	496.85	779.22	779.22	779.32	0.39	661.42
04132012PMgroup2	2.11	2.11	2.11	2.11	0.00	12.22	12.53	12.53	12.53	12.53	0.01	0.27
04132012PMgroup3	10.19	9.81	9.93	9.93	0.00	92.02	2755.97	4014.95	4014.98	4015.03	0.59	3915.28

♣Master problem was not solved to optimality at the root node so LP root node entry is not a lower bound.

♠Best IP solution is not proven optimal.

Table 3.14. Results for B&P&C components for real instances

Instance	B&B	D-W		Cuts	
	No. nodes	No. iterations	No. columns	No. cuts	Total time (sec)
11182011AMgroup1	1	19	10234	14	0.77
11182011AMgroup2	1	13	3432	2	0.06
11182011AMgroup3	1	6	247	0	0.00
01122012AMgroup1	5	48	29251	8	0.77
01122012AMgroup2	7	52	23424	42	1.01
01122012AMgroup3	1	16	1332	8	0.01
01272012AMgroup1	3	28	13575	24	2.44
01272012AMgroup2	1	12	6008	1	0.03
01272012AMgroup3	5	35	4964	15	0.12
01272012PMgroup1	1	30	7595	0	0.00
01272012PMgroup2	1	15	1668	1	0.00
01272012PMgroup3	2	46	12586	38	1.77
03052012AMgroup1	1	75	12149	0	0.00
03052012AMgroup2	1	15	1837	2	0.03
03052012AMgroup3	1	10	382	1	0.00
03052012PMgroup1	2	60	7660	7	0.51
03052012PMgroup2	1	16	3697	0	0.00
03052012PMgroup3	1	33	8110	0	0.00
04132012AMgroup1	13	93	25699	41	0.89
04132012AMgroup2	1	10	893	2	0.01
04132012AMgroup3	3	17	1738	4	0.01
04132012PMgroup1	1	27	3406	12	0.07
04132012PMgroup2	1	7	119	0	0.00
04132012PMgroup3	1	23	5626	11	0.17

Table 3.15. Performance of B&P&C algorithm for randomly generated instances

	Objective					CPU (sec)						
Instance	Heuristic	LP root node w/o cuts	LP root node	Best IP	Gap (%)	Heuristic	Root node w/o cuts	Root node	Best IP	Total	MP	SP
T2001	14.52	14.18	14.44	14.50	0.42	142.54	152.30	155.18	155.31	169.14	0.17	23.10
T2002	13.26	13.26	13.26	13.26	0.00	157.40	172.00	172.00	172.00	172.02	0.06	13.59
T2003	16.25	16.25	16.25	16.25	0.00	140.60	157.37	157.37	157.38	157.41	0.10	14.01
T2004	14.52	14.48	14.52	14.52	0.00	150.35	158.88	161.16	161.16	161.18	0.06	9.86
T2005	16.28	16.00	16.28	16.28	0.00	147.84	160.74	168.38	168.38	168.41	0.13	19.15
T3001	18.38	17.79	18.03	18.04	0.07	211.22	264.73	292.93	296.47	632.21	0.71	403.69
T3002	21.16	20.93	21.09	21.16	0.33	259.27	305.33	323.92	325.64	481.21	0.53	211.02
T3003	20.40	20.26	20.40	20.40	0.00	224.12	313.89	329.55	329.55	329.59	0.24	101.79
T3004	19.16	18.78	19.14	19.16	0.10	243.58	284.72	328.77	330.83	621.66	0.65	364.71
T3005	19.90	19.57	19.60	19.61	0.03	228.88	275.56	286.75	288.51	450.00	0.52	209.49
T4001	23.74	23.43	23.46	23.52	0.25	332.65	553.36	620.25	623.27	2947.37	1.43	2569.30
T4002	25.21	24.95	25.20	25.20	0.00	467.91	600.76	673.97	673.97	674.03	0.55	196.53
T4003	26.10	25.75	26.06	26.06	0.00	406.63	480.43	578.60	578.60	578.69	0.53	161.81
T4004	27.82	27.28	27.36	27.36	0.00	339.32	468.15	505.58	505.58	505.66	0.49	156.34
T4005	26.99	26.19	26.48	26.83	1.33	401.77	553.80	616.34	623.46	1425.35	1.47	981.49
T5001	27.12	26.28	26.48	26.83♠	1.36	310.48	1087.18	1561.64	1578.52	11297.21	6.60	10809.07
T5002	28.12	27.23	27.94	27.95♠	0.03	358.07	1402.30	1940.18	1951.30	12821.26	5.50	12407.27
T5003	29.63	28.66	28.95	29.02	0.23	297.18	959.09	1290.77	1296.86	3002.63	3.30	2616.08
T5004	29.75	28.64	28.87	28.87	0.00	412.59	895.80	1065.80	1065.80	1065.92	1.00	639.95
T5005	31.59	29.39	29.92	30.13	0.72	249.55	691.37	1813.31	1820.83	2721.07	2.88	2412.68

♠Best IP solution is not proven optimal.

Table 3.16. Results for B&P&C components for randomly generated instances

	B&B	D-W		Cuts	
Instance	No. nodes	No. iterations	No. columns	No. cuts	Total time (sec)
T2001	3	28	4292	12	0.17
T2002	1	14	1345	0	0.00
T2003	1	10	3169	0	0.00
T2004	1	12	1250	2	0.03
T2005	1	17	1496	16	0.11
T3001	3	33	9935	25	1.26
T3002	3	35	8413	7	0.66
T3003	1	19	3178	5	0.22
T3004	3	33	8309	24	1.44
T3005	3	34	8925	15	0.73
T4001	7	66	26238	10	3.23
T4002	1	24	5554	13	2.00
T4003	1	19	5435	19	2.44
T4004	1	18	4109	19	4.33
T4005	5	57	16943	37	10.51
T5001	8	125	47412	115	62.46
T5002	2	129	14034	23	12.20
T5003	5	74	19047	114	50.26
T5004	1	31	5405	17	3.94
T5005	3	99	7565	106	35.62

Table 3.17. Performance of B&P&C algorithm for real instances with nonconfigurable vehicles

	Objective					CPU (sec)						
Instance	Heuristic	LP root node w/o cuts	LP root node	Best IP	Gap (%)	Heuristic	Root node w/o cuts	Root node	Best IP	Total	MP	SP
11182011AMgroup1	13.56	13.35	13.48	13.48	0.00	73.60	460.30	816.35	816.36	816.42	0.67	729.82
11182011AMgroup2	16.15	16.06	16.11	16.11	0.00	96.94	106.43	110.12	110.12	110.13	0.06	12.32
11182011AMgroup3	3.68	3.63	3.63	3.63	0.00	9.18	9.39	9.39	9.39	9.39	0.01	0.18
01122012AMgroup1	14.25	13.95	13.99	14.00	0.08	187.85	592.18	712.24	1811.05	2994.41	2.09	2737.32
01122012AMgroup2	10.08	9.87	10.01	10.03	0.16	141.40	185.60	211.63	214.66	563.63	1.35	390.19
01122012AMgroup3	7.78	7.65	7.75	7.75	0.00	84.26	103.58	112.61	112.62	112.64	0.08	27.34
01272012AMgroup1	12.11	12.00	12.08	12.09	0.03	131.22	254.48	524.61	527.09	1120.64	0.81	971.69
01272012AMgroup2	10.60	10.51	10.54	10.54	0.00	154.15	183.26	188.58	188.58	188.60	0.14	31.46
01272012AMgroup3	7.58	7.54	7.56	7.57	0.16	77.79	83.11	86.43	86.46	106.07	0.12	26.80
01272012PMgroup1	9.86	9.74	9.77	9.77	0.00	75.81	2120.34	3108.87	3110.76	12389.40	1.02	12304.16
01272012PMgroup2	3.35	3.08	3.13	3.13	0.00	60.20	86.27	100.49	100.49	100.51	0.25	39.34
01272012PMgroup3	12.97	12.07	12.23	12.30♠	0.59	81.29	1784.47	4270.02	4273.75	11374.28	1.06	11277.52
03052012AMgroup1	18.25	17.41	17.41♣	17.56♠	0.89	99.83	10978.77	10978.77	10986.05	10986.95	3.11	10862.77
03052012AMgroup2	10.93	10.86	10.90	10.90	0.00	92.46	109.28	111.45	111.45	111.47	0.06	17.98
03052012AMgroup3	5.66	5.59	5.64	5.64	0.00	32.71	34.05	34.14	34.14	34.14	0.02	1.33
03052012PMgroup1	11.43	11.00	11.03	11.10	0.67	101.26	3008.94	4218.23	4222.77	12389.51	1.31	12268.21
03052012PMgroup2	5.31	5.29	5.29	5.29	0.00	72.79	601.82	601.84	601.84	601.92	0.16	526.20
03052012PMgroup3	13.59	13.04	13.04♣	13.16♠	0.95	125.75	13382.55	13382.55	13389.53	13391.46	0.68	13246.51
04132012AMgroup1	9.70	9.56	9.58	9.68	1.05	61.31	76.67	83.29	83.48	212.50	0.30	146.18
04132012AMgroup2	7.10	7.05	7.07	7.07	0.00	42.92	44.64	44.83	44.83	44.83	0.02	1.72
04132012AMgroup3	3.85	3.80	3.83	3.85	0.49	16.85	17.56	17.80	17.83	19.40	0.03	2.25
04132012PMgroup1	7.70	7.69	7.69	7.69	0.00	60.66	122.66	122.67	122.67	122.68	0.03	61.23
04132012PMgroup2	2.11	2.11	2.11	2.11	0.00	7.39	7.63	7.63	7.63	7.63	0.01	0.19
04132012PMgroup3	10.05	9.90	9.93	9.93	0.00	62.66	1521.19	2211.06	2211.06	2211.09	0.20	2145.79

♣Master problem was not solved to optimality at the root node so LP root node entry is not a lower bound.

♠Best IP solution is not proven optimal.

Table 3.18. Results for B&P&C components for real instances with nonconfigurable vehicles

Instance	B&B	D-W		Cuts	
	No. nodes	No. iterations	No. columns	No. cuts	Total time (sec)
11182011AMgroup1	1	13	6541	25	2.13
11182011AMgroup2	1	11	1040	2	0.06
11182011AMgroup3	1	6	91	0	0.00
01122012AMgroup1	5	48	29251	8	0.99
01122012AMgroup2	7	52	23424	42	1.27
01122012AMgroup3	1	16	1134	8	0.01
01272012AMgroup1	5	43	8730	48	5.71
01272012AMgroup2	1	12	3283	1	0.03
01272012AMgroup3	5	36	2071	20	0.20
01272012PMgroup1	2	57	6013	4	0.25
01272012PMgroup2	1	9	991	2	0.01
01272012PMgroup3	2	53	8385	21	0.94
03052012AMgroup1	1	68	6961	0	0.00
03052012AMgroup2	1	12	1312	2	0.04
03052012AMgroup3	1	9	234	1	0.00
03052012PMgroup1	3	59	11350	4	0.13
03052012PMgroup2	1	16	3697	0	0.00
03052012PMgroup3	1	28	8179	0	0.00
04132012AMgroup1	9	56	7544	38	0.59
04132012AMgroup2	1	9	387	2	0.01
04132012AMgroup3	3	14	805	4	0.00
04132012PMgroup1	1	8	1230	0	0.00
04132012PMgroup2	1	6	119	0	0.00
04132012PMgroup3	1	22	2863	4	0.07

3.7 SUMMARY AND CONCLUSIONS

In this chapter, we introduced a variant on the PDP with time windows in which the fleet is nonhomogeneous and there are multiple types of demand. An additional feature of the problem not previously addressed is that the configuration of each vehicle, and hence its ability to handle the different demand types, is part of the decision process rather than an input parameter. Solutions were found with a tailored multi-start adaptive large neighborhood search algorithm to eight real instances provided by a senior activity center and compared with the manual solutions currently being used. On average, MSALNS achieved cost saving in the range of 30 to 40% within 35 to 50 minutes, a short enough time to be used daily to construct efficient routes that take participant satisfaction into account. The algorithm can also be used for long-term planning to estimate the size and composition of the fleet required to accommodate future demand. Another advantage of the approach is its inherent flexibility. Should management's priorities change, for example, it is straightforward to adjust the objective function coefficients accordingly. In a similar vein, additional objectives can be incorporated with little effort. A final point concerns speeding up the computations. If faster run times are required, it is an easy matter to parallelize each of the two phases that constitute the MSALNS.

For cases where time windows are more restricted, we developed a B&P&C algorithm that is capable of routinely solving instances with up to 50 clients. For the larger instances that could not be solved to optimality in the 3-hour time limit, the algorithm can work as a heuristic to improve the solutions obtained with our ALNS procedure. At each Dantzig-Wolfe iteration, the elementary shortest path subproblems were solved with an enhanced labeling algorithm. Efficient dominance conditions were developed to speed up the computations. Detailed implementation aspects of the labeling algorithm are discussed along with the use of subset-row inequalities in the master problem to tighten its lower bound. The results showed that these cuts can be generated quickly and provided a noticeable improvement in the effectiveness of the algorithm without increasing runtimes. In fact, the vast majority of time consumed by the B&P&C algorithm is spent solving the subproblems with the labeling algorithm. This suggests that a worthwhile area for future research would involve the design of other exact methods or heuristics for solving the subproblem more efficiently.

A final contribution of this work concerns the construction and distribution of a set of test instances for use by the research community. Many sophisticated algorithms for the PDP have been proposed over the years, and it is likely that any number of them could be adapted to solve the variant described in this chapter.

Chapter 4. Summary and Future Research

In this dissertation, we investigated two variants of the PDP derived from real life applications. One is the PDPT where transshipment locations can be used to transfer customer shipments from one vehicle to another vehicle. The other is the HPDPCC in which vehicle capacity configurations can be adjusted to fit the need of different customer demands. For the PDPT, we implement a branch and price algorithm which turned out to be effective only for small size instances. Therefore we introduce a GRASP using ALNS as the improvement algorithm. The GRASP was proven to be effective by testing against well-known benchmarks and generated instances with known optimal solutions. For the HPDPCC, we developed an MSALNS algorithm. It was shown to be able to find solutions that provided 30%-40% improvement over current practice. We also introduced a branch and price and cut algorithm which can solve most real life instance with up to 50 customers under more restricted time window setting. In the following sections, we provide a summary of our heuristic solution methods and column generation based exact solution methods, respectively. We also point out potential areas for future research.

4.1 HEURISTIC SOLUTION METHODS

Our heuristics contain two phases. In Phase I, we start with a set of initial solutions that are created using randomized construction methods. In GRASP for the PDPT, we construct each solution by sequentially inserting customer requests into existing routes. In each step, a candidate list is created by using the best insertion positions. An insertion is then randomly selected from the candidate list and used to extend the current partial solution. In MSALNS for the HPDPCC, we randomly select from a set of ranking algorithms and similarly extend the partial solution sequentially in accordance with the algorithm selected. In each step, a customer request is inserted at the best insertion position. In both construction methods, we include randomized components in order to generate a diversified population of solutions. In Phase II, we use local search to further improve the solutions obtained in Phase I. It may not be practical to perform local search on each Phase I solution, however, so we solve a max diversity problem in MSALNS for the HPDPCC to identify a set of solutions that are most "diversified" from the initial set of Phase I solutions. The definition of "diversified" is problem specific; the idea is to start Phase II from drastically different starting solutions, so there is a better chance of finding a global minimum.

In Phase II, we use a ruin and recreate paradigm to reconstruct part of a solution. At each iteration, a percentage of customer requests are removed from the current solution and reinserted back to construct a new solution. The size of the reconstructive neighborhood grows exponentially with the increase in the number requests being removed, which is a percentage of total customer requests in the problem, which implies that the neighborhood size increases exponentially when the problem size increases. This type of local search algorithm is called large neighborhood search (LNS). LNS uses one of the removal heuristics to remove requests from the current solution, then uses one of the insertion heuristics to reinsert the requests back to reconstruct a new solution. The probability of each removal or insertion heuristic being picked at each iteration is proportional to the relative weights of these heuristics. As part of the learning mechanism, these weights are dynamically adjusted based on the quality of the solution that each provides. Another learning mechanism in our algorithms adaptively adjusts the portion of requests being removed at each iteration. The heuristic gauges the percentage of customers to remove based on the solutions obtained in a certain period. If the same set of local minima are visited repeatedly, the percentage to remove is increased to guide the search to explore other areas of the feasible region. When a better incumbent solution is found, this percentage is reduced to intensify the search around the solution.

The heuristics we proposed can be easily applied to other variants of the PDP or VRP. One may need to find problem specific insertion and removal heuristics, but the other components of the heuristics, including the adaptive learning mechanism, can be reused.

One potential shortcoming associated with our randomized heuristics is the repetitive effort whereby the same set of removal and insertion operations may be performed multiple times on the same solution. This problem is alleviated by dynamic adjusting the removal and insertion heuristic weights. We also introduced cache structures to reduce the effort spent on the same insertion operations. One future topic is to develop more effective learning mechanisms to identify, reduce and eliminate repetitive steps.

4.2 COLUMN GENERATION BASED METHODS

In branch and price, column generation is used to solve each node in the branch and bound tree. At each Dantzig-Wolfe iteration, one or more subproblems are solved to find columns (routes in our case) with negative reduced costs with respect to the restricted master problem. The column

generation stops when there is no column found by the subproblems. If the solution at the node is not integral, valid inequalities can be added to the restricted master problem to tighten the formulation. Once valid inequalities are added to the restricted master problem, Dantzig-Wolfe iterations are performed until no column with a negative reduced cost is found. The process of adding cuts is repeated until no more cuts or columns can be found, at which time, branch and bound is applied.

In our work, elementary shortest path subproblems were solved with a labeling algorithm. Because the majority of the computation time is spent solving the subproblems, it is critical that they are solved quickly. Effective labeling algorithms require efficient dominance conditions to reduce the solution space that needs to be explicitly explored. As part of the research, efficient dominance conditions are developed for the HPDPCC that make use of the property of the delivery triangle inequality. Improving the strength of the dominance conditions for the PDPT is an area of future research.

Another way we found to improve the efficiency of branch and price is to perform local search on solution columns and columns generated by previous iterations. Although labeling algorithms have been applied extensively in the literature, there has been relatively less effort devoted to discussing the details of implementation, such as how the graph is traversed, or when and how the dominance conditions are examined. In B&P&C for the HPDPCC, we described a labeling algorithm that traverses the graph using a priority queue. Our priority rule definition balances the effectiveness of the dominance conditions with the memory footprint that the algorithm consumes. Although this labeling algorithm implementation turns out to be effective for our problem, it would be interesting to see if more sophisticated graph traversal methods can further improve performance.

In B&P&C for the HPDPCC, we introduced subset-row inequalities in the master problem to tighten its lower bound. The results showed that these cuts can be generated quickly and provide a noticeable improvement in the effectiveness of the algorithm. It would be interesting to identify and evaluate other valid inequalities to further improve the algorithms.

Baldacci et al. (2011) developed an exact algorithm for the PDP in which they introduced bounding procedures to find good dual solutions for the relaxed master problem. By coupling dual solutions with known upper bound, they were able to substantially reduce the number of

variables considered in the master problem. It is worth investigating the possibility and effectiveness applying similar methods to our problems.

Overall, the column generation based exact solution method proved to be effective in finding optimal solutions mainly for PDPs with restricted time windows. The loosely constrained problems remain challenging. Although we have seen that these methods can be easily adapted to work as heuristics, especially when integrated with other heuristics, column generation based methods can further improve the solutions obtained by these heuristics.

Appendices

APPENDIX A: PARAMETER VALUES USED IN TESTING FOR GRASP FOR PDPT

The parameter settings used for the Li and Lim (2001) PDP data sets are given below. When different, the settings for the PDPT data sets are given in parentheses.

- Number of GRASP iterations, n_I^{max} = 400 (120)
- Total time out limit for Phase II iterations, $n_I^{max} \cdot$ Phase II frequency $\cdot T_{II}^{max}$ = 600 (1200) sec
- Maximum number of Phase II iterations per call, n_{II}^{max} = 50,000
- Weights used to determine the cost $COST(S)$ of a solution S, $w_1 = 1$, $w_2 = 10000$, $w_3 = 1{,}000{,}000$
- Initial weight for Shaw removal heuristic, $rw_1 = 30$ Initial weight for random removal heuristic, $rw_2 = 30$
- Initial weight for random route removal heuristic, $rw_3 = 30$
- Initial weight for greedy insertion heuristic, $ih_1 = 20$
- Initial weight for regret-2 insertion heuristic, $ih_2 = 20$
- Initial weight for random insertion heuristic, $ih_3 = 20$
- Initial weight for most constrained insertion heuristic $ih_4 = 20$
- Weights used to determine the relatedness measure, $\theta_1 = 0.2$, $\theta_2 = 0.1$, $\theta_3 = 1.0$
- Parameter used to randomly pick a request to remove in Shaw removal heuristic, $\alpha = 5$
- Parameters to determine the relatedness measure, $\beta_1 = 0.5$, $\beta_2 = 0.2$, $\beta_3 = 0.3$
- Parameter used to randomly pick a request to remove in most constrained first insertion heuristic, $\beta = 5$
- Probability to disable single route insertion for `Basic_Insertion` procedure, $\gamma = 0$ (0.2)
- Number of iterations in a segment = 100
- Parameters to determine the scores of heuristics in a segment, $\delta_1 = 30$, $\delta_1 = 3$
- The reaction factor to determine the rate of weight changes for heuristics, $\rho = 0.1$

- Parameters to determine the number of shipment requests to remove in each Phase II iteration, $m = 3$, $\mu_1 = 5$, $\mu_2 = \mu_a = 10$ (12), $\mu_3 = 15$ (20)
- Maximum number of repeated solutions before increasing the number of requests μ to be removed, $\lambda_1 = 14$
- Number of iterations in which the same solution is repeated before the number of shipment requests to be removed is incremented, $\lambda_2 = 20$

APPENDIX B: BEST KNOWN SOLUTIONS OF EXISTING LI&LIM DATA SET

The following tables contain the best known objective function values for the Li and Lim (2001) data sets. We were able to achieve these values in all cases but not always with the same parameter settings.

Table B1. Data set LC1

Test case	Number of vehicles used	Best known total travel distance
1	10	828.94
2	10	828.94
3	9	1035.35
4	9	860.01
5	10	828.94
6	10	828.94
7	10	828.94
8	10	826.44
9	9	1000.60

Table B2. Data set LC2

Test case	Number of vehicles used	Best known total travel distance
1	3	591.56
2	3	591.56
3	3	591.17
4	3	590.60
5	3	588.88
6	3	588.49
7	3	588.29
8	3	588.32

Table B3. Data set LR1

Test case	Number of vehicles used	Best known total travel distance
1	19	1650.8
2	17	1487.57
3	13	1292.68
4	9	1013.39
5	14	1377.11
6	12	1252.62
7	10	1111.31
8	9	968.97
9	11	1208.96
10	10	1159.35
11	10	1108.90
12	9	1003.77

Table B4. Data set LR2

Test case	Number of vehicles used	Best known total travel distance
1	4	1253.23
2	3	1197.67
3	3	949.4
4	2	849.05
5	3	1054.02
6	3	931.63
7	2	903.06
8	2	734.85
9	3	930.59
10	3	964.22
11	2	911.52

Table B5. Data set LRC1

Test case	Number of vehicles used	Best known total travel distance
1	14	1708.8
2	12	1558.07
3	11	1258.74
4	10	1128.4
5	13	1637.62
6	11	1424.73
7	11	1230.14
8	10	1147.43

Table B6. Data set LRC2

Test case	Number of vehicles used	Best known total travel distance
1	4	1406.94
2	3	1374.27
3	3	1089.07
4	3	822.34
5	4	1302.2
6	3	1159.03
7	3	1062.05
8	3	852.76

APPENDIX C: OPTIMAL SOLUTIONS FOR GENERATED DATA SETS FOR PDPT

The input data are available at: https://webspace.utexas.edu/quyuan/public/pdpt/

Table C1. Optimal solutions for randomly generated data set

Test case with transshipment location	Number of vehicles used	Optimal total travel distance
pdpt1	2	2510.94
pdpt2	2	2359.01
pdpt3	2	2384.84
pdpt4	2	2405.80
pdpt5	3	3479.03

APPENDIX D: INPUT DATA FORMAT AND SAMPLES FOR PDPT

There is one input file for each PDPT instance, the input file format is similar as the ones for Li and Lim (2001) data sets. The input files are in text format as shown in Table D1, where entries in a row are separated by tabs.

Table D1. Sample input file from Li&Lim without transshipment

Number of vehicles	Vehicle capacity	Vehicle speed						
Customer request number	X coordinate	Y coordinate	Load	Earliest service start time	Latest service start time	Service time	Pickup customer request number	Delivery customer request number
…	…	…	…	…	…	…	…	…

The first row of the file contains vehicle information, including number of vehicles, vehicle capacity and vehicle speed. The second row contains the depot information, in this row, column 2 and 3 indicate the location of the depot; column 5 and 6 specify the opening and closing time of the depot. The following rows are customer information. In each row, column 1 indicates the customer number; column 2 and 3 indicates the coordinates of the service location; column 4 specifies the load associated with the customer; column 5 and 6 indicate the window in which service can be started; column 7 specifies the service time. Column 8 indicates the corresponding pickup customer request number, if this column is not 0, then it indicate that the customer request specified in this row is a delivery customer request whose corresponding pickup request is specified in this column. Similarly column 9 indicates the corresponding delivery customer request number. The rows at the end contain transshipment information if either column 8 or 9 contains –1. For these rows, column 2 and 3 indicate the location of the transshipment facility; column 5 and 6 specify the opening and closing time of the transshipment facility.

Table D2. Sample input file from Li&Lim without transshipment

4	1000	1						
0	40.00	50.00	0	0	960	0	0	0
1	25.00	85.00	-10	673	793	10	14	0
2	22.00	75.00	30	152	272	10	0	98
3	22.00	85.00	10	471	591	0	0	101
4	20.00	80.00	-19	644	764	10	59	0
5	20.00	85.00	20	73	193	10	0	88
6	18.00	75.00	-20	388	508	10	12	0
7	15.00	75.00	-16	300	420	10	65	0
8	15.00	80.00	10	367	487	10	0	46
9	10.00	35.00	-13	371	491	10	87	0

10	10.00	40.00	-3	519	639	10	62	0
11	8.00	40.00	40	195	315	10	0	68
12	8.00	45.00	20	223	343	10	0	6
13	5.00	35.00	-23	653	773	10	57	0
14	5.00	45.00	10	35	155	10	0	1
15	2.00	40.00	20	174	294	10	0	55
16	0.00	40.00	20	255	375	10	0	73
17	0.00	45.00	20	703	823	10	0	100
18	44.00	5.00	-9	335	455	10	82	0
19	42.00	10.00	40	254	374	10	0	51
20	42.00	15.00	10	537	657	10	0	54
21	40.00	5.00	-30	215	335	10	23	0
22	40.00	15.00	-2	375	495	10	85	0
23	38.00	5.00	30	201	321	10	0	21
24	38.00	15.00	-25	681	801	10	89	0
25	35.00	5.00	-10	784	904	10	48	0
26	95.00	30.00	-20	529	649	10	41	0
27	95.00	35.00	20	146	266	10	0	99
28	92.00	30.00	10	149	269	10	0	67
29	90.00	35.00	10	194	314	10	0	30
30	88.00	30.00	-10	246	366	10	29	0
31	88.00	35.00	20	165	285	10	0	90
32	87.00	30.00	-30	621	741	10	38	0
33	85.00	25.00	10	80	200	10	0	71
34	85.00	35.00	-10	487	607	10	39	0
35	67.00	85.00	-3	657	777	10	52	0
36	65.00	85.00	40	43	163	10	0	72
37	65.00	82.00	10	557	677	10	0	80
38	62.00	80.00	30	278	398	10	0	32
39	60.00	80.00	10	64	184	10	0	34
40	60.00	85.00	-10	329	449	10	45	0
41	58.00	75.00	20	376	496	10	0	26
42	55.00	80.00	10	33	153	10	0	44
43	55.00	85.00	-17	574	694	10	64	0
44	55.00	82.00	-10	217	337	10	42	0
45	20.00	82.00	10	37	157	10	0	40
46	18.00	80.00	-10	489	609	10	8	0
47	2.00	45.00	10	105	225	10	0	78
48	42.00	5.00	10	732	852	10	0	25
49	42.00	12.00	10	440	560	10	0	56
50	72.00	35.00	-10	507	627	10	81	0
51	55.00	20.00	-40	326	446	10	19	0

52	25.00	30.00	3	175	295	10	0	35
53	20.00	50.00	-14	375	495	10	92	0
54	55.00	60.00	-10	601	721	10	20	0
55	30.00	60.00	-20	599	719	10	15	0
56	50.00	35.00	-10	557	677	10	49	0
57	30.00	25.00	23	397	517	10	0	13
58	15.00	10.00	-7	782	902	10	77	0
59	10.00	20.00	19	42	162	10	0	4
60	15.00	60.00	-35	694	814	10	97	0
61	45.00	65.00	9	258	378	10	0	94
62	65.00	35.00	3	167	287	10	0	10
63	65.00	20.00	-6	39	159	10	95	0
64	45.00	30.00	17	191	311	10	0	43
65	35.00	40.00	16	11	131	10	0	7
66	41.00	37.00	16	566	686	10	0	93
67	64.00	42.00	-10	268	388	10	28	0
68	40.00	60.00	-40	612	732	10	11	0
69	31.00	52.00	27	157	277	10	0	76
70	35.00	69.00	23	810	930	0	0	102
71	65.00	55.00	-10	241	361	10	33	0
72	63.00	65.00	-40	60	180	10	36	0
73	2.00	60.00	-20	286	406	10	16	0
74	20.00	20.00	-28	645	765	10	86	0
75	5.00	5.00	16	232	352	10	0	79
76	60.00	12.00	-27	268	388	10	69	0
77	23.00	3.00	7	764	884	10	0	58
78	8.00	56.00	-10	365	485	10	47	0
79	6.00	68.00	-16	352	472	10	75	0
80	47.00	47.00	-10	822	942	10	37	0
81	49.00	58.00	10	355	475	10	0	50
82	27.00	43.00	9	152	272	10	0	18
83	37.00	31.00	14	105	225	10	0	84
84	57.00	29.00	-14	395	515	10	83	0
85	63.00	23.00	2	344	464	10	0	22
86	21.00	24.00	28	349	469	10	0	74
87	12.00	24.00	13	359	479	10	0	9
88	24.00	58.00	-20	260	380	10	5	0
89	67.00	5.00	25	713	833	10	0	24
90	37.00	47.00	-20	359	479	10	31	0
91	49.00	42.00	-26	719	839	10	96	0
92	53.00	43.00	14	14	134	10	0	53
93	61.00	52.00	-16	808	928	10	66	0

94	57.00	48.00	-9	392	512	10	61	0
95	56.00	37.00	6	100	220	10	0	63
96	55.00	54.00	26	562	682	10	0	91
97	4.00	18.00	35	547	667	10	0	60
98	26.00	52.00	-30	172	292	10	2	0
99	26.00	35.00	-20	308	428	10	27	0
100	31.00	67.00	-20	810	930	10	17	0
101	22.00	85.00	-10	471	591	10	3	0
102	35.00	69.00	-23	810	930	10	70	0

Table D3. Sample input file with one transshipment location

3	100	1						
0	41.00	67.00	0	0	1440	0	0	0
1	64.00	248.00	72	101	1111	0	0	2
2	105.00	145.00	72	114	836	0	1	0
3	221.00	316.00	42	321	726	0	0	4
4	144.00	39.00	42	156	838	0	3	0
5	331.00	308.00	82	222	1081	0	0	6
6	190.00	242.00	82	396	1009	0	5	0
7	341.00	111.00	35	44	852	0	0	8
8	41.00	67.00	35	66	1259	0	7	0
9	253.00	68.00	79	310	1089	0	0	10
10	278.00	158.00	79	154	1145	0	9	0
11	84.00	354.00	65	25	814	0	0	12
12	195.00	342.00	65	202	1037	0	11	0
13	335.30	118.03	16	87	537	0	0	14
14	231.60	256.48	16	192	781	0	13	0
15	335.30	118.03	14	87	537	0	0	16
16	229.38	268.95	14	199	799	0	15	0
17	41.77	73.09	12	0	108	0	0	18
18	195.00	342.00	12	202	1037	0	17	0
19	326.93	128.36	16	95	555	0	0	20
20	198.94	127.65	16	575	1440	0	19	0
21	231.60	256.48	4	192	781	0	0	22
22	131.04	101.57	4	619	1440	0	21	0
23	102.87	166.20	2	88	540	0	0	24
24	195.00	342.00	2	202	1037	0	23	0
25	137.03	154.57	19	551	1440	0	0	26
26	41.00	67.00	19	629	1440	0	25	0

27	64.00	248.00	6	101	1111	0	0	28
28	153.43	346.50	6	243	902	0	27	0
29	226.70	284.00	15	208	820	0	0	30
30	182.79	178.55	15	313	1065	0	29	0
31	195.00	342.00	2	202	1037	0	0	32
32	293.07	290.24	2	378	1215	0	31	0
33	231.60	256.48	19	192	781	0	0	34
34	193.17	215.87	19	290	1011	0	33	0
35	115.35	350.61	15	220	848	0	0	36
36	192.54	242.06	15	448	1378	0	35	0
37	144.00	39.00	9	156	838	0	0	38
38	187.97	50.70	9	428	1332	0	37	0
39	137.03	154.57	9	551	1440	0	0	40
40	41.00	67.00	9	629	1440	0	39	0
41	230.40	139.72	1	555	1440	0	0	42
42	198.94	127.65	1	575	1440	0	41	0
43	278.00	158.00	13	154	1145	0	0	44
44	131.04	101.57	13	619	1440	0	43	0
45	177.48	191.46	3	518	1440	0	0	46
46	41.00	67.00	3	66	1259	0	45	0
47	56.47	69.27	2	0	121	0	0	48
48	221.00	316.00	2	321	726	0	47	0
49	230.40	139.72	15	555	1440	0	0	50
50	131.04	101.57	15	619	1440	0	49	0
51	234.00	243.00	0	0	1440	0	-1	-1

APPENDIX E: INPUT DATA FORMAT AND SAMPLES FOR HPDPCC

Table E1. Sample input file for vehicle information

Vehicle name	Count	Speed (miles/min)	Depot			Window		Capacity configurations			
			name	latitude	longitude	Start	end	seat1	wheelchair1	seat2	wheelchair2
VAN1	1	0.5	HOPE	37.6949	-97.3729	7:00AM	12:30PM	1	0	0	1
VAN2	3	0.5	HOPE	37.6949	-97.3729	7:00AM	12:30PM	8	2		
VAN3	6	0.5	HOPE	37.6949	-97.3729	7:00AM	12:30PM	8	2	6	3
VAN4	1	0.5	HOPE	37.6949	-97.3729	7:00AM	12:30PM	8	2	4	4

Table E2. Sample input file of a generated instance for HPDPCC with relax time windows

Cus name	Pickup information						Delivery information						Demand		
	Latitude	Longitude	Location	Start	End	Service time	Latitude	Longitude	Location	Start	End	Service time	ST	WC	WK
c0	37.6849	-97.3029	c0's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c1	37.8149	-97.1929	c1's residence	10:30	11:55	5	37.6949	-97.3729	center	10:30	11:55	1	1	0	0
c2	37.7649	-97.4629	c2's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c3	37.7849	-97.5329	c3's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c4	37.6149	-97.3629	c4's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c5	37.7049	-97.5129	c5's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c6	37.7449	-97.3829	c6's residence	7:35	8:50	5	37.6949	-97.3729	center	7:35	8:50	1	1	0	0
c7	37.6049	-97.3329	c7's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c8	37.6649	-97.5329	c8's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c9	37.7049	-97.3629	c9's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c10	37.6649	-97.2729	c10's residence	7:30	8:50	5	37.6949	-97.3729	center	7:30	8:50	1	1	0	0
c11	37.7949	-97.5129	c11's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c12	37.6549	-97.2729	c12's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c13	37.7549	-97.2829	c13's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0

c14	37.7949	-97.5729	c14's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c15	37.5949	-97.1829	c15's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c16	37.6149	-97.1929	c16's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c17	37.6249	-97.1829	c17's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c18	37.6549	-97.2429	c18's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c19	37.5949	-97.2729	c19's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c20	37.5649	-97.2529	c20's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c21	37.6649	-97.5029	c21's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c22	37.7749	-97.2729	c22's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c23	37.8249	-97.2629	c23's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c24	37.7849	-97.2429	c24's residence	10:50	12:00	5	37.6949	-97.3729	center	10:50	12:00	1	1	0	0
c25	37.7949	-97.3529	c25's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c26	37.7549	-97.3729	c26's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c27	37.5849	-97.3329	c27's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c28	37.6449	-97.5529	c28's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c29	37.7149	-97.2129	c29's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c30	37.6749	-97.5329	c30's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c31	37.6949	-97.3929	c31's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c32	37.7449	-97.4929	c32's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c33	37.6849	-97.2729	c33's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c34	37.5649	-97.4129	c34's residence	7:50	9:00	5	37.6949	-97.3729	center	7:50	9:00	1	1	0	0
c35	37.6149	-97.3429	c35's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c36	37.7249	-97.2829	c36's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c37	37.6849	-97.3429	c37's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c38	37.6649	-97.4929	c38's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c39	37.5549	-97.2129	c39's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c40	37.6449	-97.2029	c40's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c41	37.7349	-97.3829	c41's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c42	37.7449	-97.4929	c42's residence	7:50	9:00	5	37.6949	-97.3729	center	7:50	9:00	1	1	0	0
c43	37.5549	-97.5129	c43's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0

c44	37.7649	-97.3529	c44's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c45	37.7349	-97.4629	c45's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c46	37.5549	-97.4529	c46's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c47	37.7349	-97.5429	c47's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c48	37.7249	-97.4229	c48's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c49	37.7349	-97.5229	c49's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c50	37.5949	-97.4529	c50's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c51	37.6749	-97.4829	c51's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c52	37.7349	-97.2829	c52's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c53	37.7349	-97.2429	c53's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c54	37.8249	-97.4829	c54's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c55	37.8049	-97.2229	c55's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c56	37.7149	-97.5529	c56's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c57	37.6749	-97.2629	c57's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c58	37.7649	-97.4429	c58's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c59	37.5649	-97.2429	c59's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c60	37.6549	-97.5629	c60's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c61	37.6049	-97.4229	c61's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c62	37.7049	-97.4529	c62's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c63	37.8149	-97.3229	c63's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c64	37.7749	-97.4929	c64's residence	9:15	10:25	10	37.6949	-97.3729	center	9:15	10:25	3	0	1	0
c65	37.6549	-97.3229	c65's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c66	37.7349	-97.2329	c66's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c67	37.6649	-97.4029	c67's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c68	37.6949	-97.5429	c68's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	1
c69	37.7949	-97.4429	c69's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c70	37.6249	-97.3229	c70's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c71	37.6849	-97.3629	c71's residence	7:45	8:55	10	37.6949	-97.3729	center	7:45	8:55	3	0	1	0
c72	37.6649	-97.4529	c72's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c73	37.5949	-97.3029	c73's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0

c74	37.7749	-97.2229	c74's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c75	37.7449	-97.5229	c75's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c76	37.6349	-97.3829	c76's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c77	37.6149	-97.3529	c77's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c78	37.7849	-97.5329	c78's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c79	37.7949	-97.5729	c79's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c80	37.8049	-97.4729	c80's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c81	37.5849	-97.1929	c81's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c82	37.8049	-97.5029	c82's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c83	37.5649	-97.4929	c83's residence	10:40	12:00	10	37.6949	-97.3729	center	10:40	12:00	3	0	1	0
c84	37.6749	-97.4029	c84's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
c85	37.5549	-97.5529	c85's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c86	37.5849	-97.4629	c86's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c87	37.7149	-97.2729	c87's residence	7:30	12:00	10	37.6949	-97.3729	center	7:30	12:00	3	0	1	0
c88	37.6749	-97.4629	c88's residence	10:45	12:00	5	37.6949	-97.3729	center	10:45	12:00	1	1	0	0
c89	37.7649	-97.5029	c89's residence	7:30	12:00	5	37.6949	-97.3729	center	7:30	12:00	1	1	0	0
ca90	37.6249	-97.5729	c90's residence	9:15	9:45	5	37.6349	-97.5529	appointment	9:15	10:15	1	1	0	1
cb90	37.6349	-97.5529	appointment	11:15	11:45	5	37.6249	-97.5729	c90's residence	11:15	12:15	1	1	0	1
ca91	37.6049	-97.4729	c91's residence	9:00	9:30	5	37.6399	-97.3829	appointment	9:00	10:00	1	1	0	0
cb91	37.6399	-97.3829	appointment	11:00	11:30	5	37.6049	-97.4729	c91's residence	11:00	12:00	1	1	0	0
ca92	37.5849	-97.3329	c92's residence	9:05	9:35	5	37.6249	-97.2729	appointment	9:05	10:05	1	1	0	0
cb92	37.6249	-97.2729	appointment	11:05	11:35	5	37.5849	-97.3329	c92's residence	11:05	12:05	1	1	0	0
ca93	37.5949	-97.4129	c93's residence	9:05	9:35	10	37.5849	-97.3079	appointment	9:05	10:05	3	0	1	0
cb93	37.5849	-97.3079	appointment	11:05	11:35	10	37.5949	-97.4129	c93's residence	11:05	12:05	3	0	1	0
ca94	37.5849	-97.2829	c94's residence	9:00	9:30	10	37.6549	-97.3479	appointment	9:00	10:00	3	0	1	0
cb94	37.6549	-97.3479	appointment	11:00	11:30	10	37.5849	-97.2829	c94's residence	11:00	12:00	3	0	1	0
ca95	37.6049	-97.4129	c95's residence	9:05	9:35	10	37.6849	-97.4879	appointment	9:05	10:05	3	0	1	0

cb95	37.6849	-97.4879	appointment	11:05	11:35	10	37.6049	-97.4129	c95's residence	11:05	12:05	3	0	1	0
ca96	37.5649	-97.1829	c96's residence	7:30	8:00	5	37.5599	-97.2979	appointment	7:30	8:30	1	1	0	0
cb96	37.5599	-97.2979	appointment	9:30	10:00	5	37.5649	-97.1829	c96's residence	9:30	10:30	1	1	0	0
ca97	37.8149	-97.4329	c97's residence	9:15	9:45	5	37.7049	-97.4179	appointment	9:15	10:15	1	1	0	0
cb97	37.7049	-97.4179	appointment	11:15	11:45	5	37.8149	-97.4329	c97's residence	11:15	12:15	1	1	0	0
ca98	37.7449	-97.2629	c98's residence	7:45	8:15	5	37.7249	-97.2829	appointment	7:45	8:45	1	1	0	0
cb98	37.7249	-97.2829	appointment	9:45	10:15	5	37.7449	-97.2629	c98's residence	9:45	10:45	1	1	0	0
ca99	37.7749	-97.2329	c99's residence	9:05	9:35	5	37.7399	-97.2979	appointment	9:05	10:05	1	1	0	0
cb99	37.7399	-97.2979	appointment	11:05	11:35	5	37.7749	-97.2329	c99's residence	11:05	12:05	1	1	0	0

Table E3. Sample input file of a generated instance for HPDPCC with customer selected pickup time slots

Cus name	Pickup information						Delivery information						Demand		
	Latitude	Longitude	Location	Start	End	Service time	Latitude	Longitude	Location	Start	End	Service time	ST	WC	WK
c0	37.6349	-97.3329	c0's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0
c1	37.6049	-97.3129	c1's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c2	37.6449	-97.3429	c2's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
c3	37.8049	-97.3929	c3's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	1
c4	37.6549	-97.4629	c4's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0
c5	37.8049	-97.5429	c5's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c6	37.7449	-97.3129	c6's residence	7:40	7:50	10	37.6949	-97.3729	center	7:40	8:40	3	0	1	0
c7	37.5749	-97.2429	c7's residence	7:50	8:00	10	37.6949	-97.3729	center	7:50	8:50	3	0	1	0
c8	37.7849	-97.4029	c8's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	1
c9	37.5849	-97.2329	c9's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c10	37.6649	-97.4229	c10's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0
c11	37.8149	-97.2029	c11's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0

c12	37.5849	-97.5529	c12's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
c13	37.7549	-97.3229	c13's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
c14	37.5749	-97.3529	c14's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c15	37.6549	-97.2529	c15's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
c16	37.5849	-97.3529	c16's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c17	37.8249	-97.4129	c17's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c18	37.6649	-97.3729	c18's residence	7:40	7:50	10	37.6949	-97.3729	center	7:40	8:40	3	0	1	0
c19	37.5849	-97.2129	c19's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c20	37.7349	-97.3429	c20's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0
c21	37.7149	-97.5529	c21's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c22	37.8249	-97.4129	c22's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c23	37.7849	-97.2329	c23's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	1
c24	37.7249	-97.3629	c24's residence	7:40	7:50	10	37.6949	-97.3729	center	7:40	8:40	3	0	1	0
c25	37.6749	-97.3229	c25's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c26	37.7649	-97.2229	c26's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	0
c27	37.6949	-97.3529	c27's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c28	37.8049	-97.3829	c28's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	0
c29	37.6949	-97.5229	c29's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	0
c30	37.6449	-97.3229	c30's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c31	37.5749	-97.3629	c31's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c32	37.7249	-97.2329	c32's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	1
c33	37.7549	-97.5529	c33's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c34	37.8249	-97.3029	c34's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c35	37.8149	-97.3329	c35's residence	7:30	7:40	5	37.6949	-97.3729	center	7:30	8:30	1	1	0	1
c36	37.5849	-97.4529	c36's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	0
c37	37.8149	-97.3729	c37's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c38	37.7949	-97.4729	c38's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c39	37.6949	-97.4829	c39's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	1
c40	37.6349	-97.2629	c40's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	1
c41	37.8249	-97.4729	c41's residence	7:50	8:00	5	37.6949	-97.3729	center	7:50	8:50	1	1	0	0

c42	37.5849	-97.5429	c42's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
c43	37.5549	-97.4329	c43's residence	7:40	7:50	5	37.6949	-97.3729	center	7:40	8:40	1	1	0	0
c44	37.7249	-97.4429	c44's residence	7:30	7:40	10	37.6949	-97.3729	center	7:30	8:30	3	0	1	0
ca45	37.7649	-97.3229	c45's residence	7:40	7:50	5	37.6599	-97.4079	appointment	7:45	8:45	1	1	0	1
ca46	37.5849	-97.3029	c46's residence	7:40	7:50	5	37.7049	-97.2779	appointment	7:45	8:45	1	1	0	0
ca47	37.7549	-97.2529	c47's residence	7:30	7:40	5	37.7549	-97.2379	appointment	7:35	8:35	1	1	0	1
ca48	37.6249	-97.5529	c48's residence	7:40	7:50	5	37.6999	-97.4879	appointment	7:45	8:45	1	1	0	0
ca49	37.7249	-97.2229	c49's residence	7:40	7:50	10	37.7599	-97.3629	appointment	7:40	8:40	3	0	1	0

References

Alexandrescu, A. (2001). *Modern C++ design: generic programming and design patterns applied,* Addison-Wesley, Boston, M.A.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications,* Prentice Hall, Englewood Cliffs, N.J.

Andrew, P. A., Cynthia, B., Keith, A. W., and Alysia, M. W. (2004). UPS optimizes its air network. *Interfaces,* 34(2a), 15-25.

Argüello, M.F., Bard, J.F. and Yu, G. (1997). Models and methods for managing airline irregular operations aircraft routing. *Operations Research in Airline Industry,* Yu, G. (ed), Kluwer Academic Publishers, Boston, MA, 1-45.

Armacost, A. P., Barnhart, C., and Ware, K. A. (2002). Composite variable formulations for express shipment service network design. *Transportation Science,* 36(2a), 1-20.

Baldacci R., Battarra M. and Vigo D. (2008). Routing a heterogeneous fleet of vehicles. In: Golden B.L., Raghavan S., Wasil E.A., editors. *The vehicle routing problem*: *latest advances and new challenges,* New York: Springer, 3-27.

Baldacci , R., Bartolini, E. and Mingozzi A. (2011). An exact algorithm for the pickup and delivery problem with time windows. *Operations Research,* 59(2), 414-426.

Baldacci, R. and Mingozzi, A. (2009). A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming,* 120(2), 347-380.

Bard, J.F. and Jarrah, A.I. (2009). Large-scale constrained clustering for rationalizing pickup and delivery operations. *Transportation Research Part B,* 43(2e), 542-561.

Bard, J.F., Kontoravdis G. and Yu G. (2002). A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science,* 36(2), 250-269.

Bard, J.F. and Nananukul, N. (2009). A two-stage supply chain planning problem with inventory routing. *Computers & Operations Research,* 37(12), 2202-2217.

Bard J.F., Shao Y. and Jarrah, A.I. (2011). A Sequential GRASP for the Therapist Routing and Scheduling Problem. Working paper. Graduate Program in Operations Research & Industrial Engineering, The University of Texas, Austin.

Bard, J.F., Shao, Y., Qi, X. and Jarrah, A.I. (2012). The traveling therapist scheduling problem. Working paper. Graduate Program in Operations Research & Industrial Engineering, The University of Texas, Austin, TX.

Bard, J.F. and Wan, L. (2006). The task assignment problem for unrestricted movement between workstation groups. *Journal of Scheduling*, 9(4), 315-342.

Bard, J.F., Yu, G. and Argüello, M.F. (2001). Optimizing aircraft routings in response to groundings and delays. *IIE Transactions on Operations Engineering*, 33(10), 931-947.

Barnhart, C., Krishnan, N., Kim, D., and Ware, K. (2002). Network design for express shipment delivery. *Computational Optimization and Applications*, 21(2c), 239-262.

Battiti, R., Tecchiolli, G. (1994). The reactive tabu search. *ORSA Journal of Computing*, 6(2b), 126-140.

Beltrami, E. J. and Bodin L.D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1), 65–94.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I. and Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *TOP*, 15(2a), 1-31.

Boudia M, Louly MAO, Prins C (2006). A reactive GRASP and path relinking for a combined production-distribution problem. *Computers & Operations Research*, 34(2k), 3402-3419.

Codato, G and Fischetti, M. (2006). Combinatorial Benders' cuts for mixed-integer linear programming. *Operations Research*, 54(2d), 756-766.

Cordeau, J.F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3), 573-586.

Cordeau, JF. and Laporte, G. (2003). The dial-a-ride problem (DARP): variants, modeling issues and algorithms. *4OR*, 1, 89-101.

Cordeau, J.F., Laporte, G., Savelsbergh, M.W.P. and Vigo, D. (2007). Vehicle routing. In: Barnhart, C., Laporte, editors. *Transportation, handbooks in operations research and management science*, Amsterdam: North-Holland, 367-428.

Cortés, C., Matamala, M. and Contardo, C. (2010). The pickup-and-delivery problem with transfers: formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(2c), 711-724.

Crainic, T. G., and Laporte, G. (1997). Planning models for freight transportation. *European Journal of Operational Research*, 97(2c), 409-438.

Dantzig, G.B. and Ramser, J.M. (1959). The truck dispatching problem. *Management Science*, 6(1), 81-91.

Dell'Amico, M., Monaci, M., Pagani, C. and Vigo, D. (2007). Heuristic approaches for the fleet size and mix vehicle routing problem with time windows. *Transportation Science*, 41(4), 516-526.

Desaulniers, G. (2010). Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, 58(1), 179-192.

Desrochers, M., Desrosiers, J. and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2), 342-354.

Desrosiers, J. and Dumas, Y. (1988). The shortest path for the construction of vehicle routes with pick-up, delivery and time constraints. *Lecture Notes in Economics and Mathematical Systems* 3*02; Advances in Optimization and Control*, H.A. Eiselt and Pederzoli (Eds.), Springler-Verlag, 144-157.

Desrosiers, J., Soumis, F., and Desrochers, M. (1984). Routing with time window with column generation. *Networks*, 14, 545-565.

Dumas, Y., Desrosiers, J. and Soumis, F. (1991). The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(2a), 7-22.

Dyer, M.E. and Wolsey, L.A. (1990). Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26(2-3), 255–270.

Eiselt, H.A., Gendreau M. and Laporte G. (1995). Arc routing problems, part I: the chinese postman problem. *Operations Research*, 43(2), 231-242.

Eiselt, H.A., Gendreau M. and Laporte G. (1995). Arc routing problems, part II: the rural postman problem. *Operations Research*, 43(3), 399-414.

Feillet, D., Dejax, P., Gendreau, M. and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3) 216–229.

Feo T.A., Venkatraman K. and Bard J.F. (1991). A GRASP for a difficult single machine scheduling problem. *Computers & Operations Research*, 18(2h), 635-643.

Gilbert, L., Suzanne, C., Philippe-Eric, L. and Hélène M. (1989). An algorithm for the design of mailbox collection routes in urban areas. *Transportation Research Part B: Methodological*, 23(4), 271-280.

Golden, B., Assad, A., Levy, L. and Gheysens, F. (1984). The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1), 49-66.

Hane, C., Barnhart, C., Johnson, E.L., Marsten, R.E., Nemhauser, G.L. and Sigismondi, G.

(1995). The fleet assignment problem: solving a large integer program. *Mathematical Programming*, 70 (2), 211-232.

Hansen P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(2c), 449-467.

Hoff A., Andersson, H., Christiansen, M., Hasle, G. and Løkketangen, A. (2010). Industrial aspects and literature survey: Fleet composition and routing. *Computers & Operations Research*, 37(12), 2041-2061.

Jarrah, A.I.Z., Yu, G., Krishnamurthy, N. and Rakshit, A. (1993). A decision support framework for airline fight cancellations and delays. *Transportation Science*, 27(3), 266-280.

Irnich, S. and Desaulniers. D. (2005). Shortest path problems with resource constraints. G. Desaulniers, J. Desrosiers, M.M. Solomon, eds. *Column Generation*. Springer, New York, 33-65.

Jepsen, M., Petersen, B., Spoorendonk, S. and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle routing problem with time windows. *Operations Research*, 56(2), 497-511.

Kliewer, N., Mellouli, T. and Suhl, L. (2006). A time–space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175 (3), 1616–1627.

Kim, D., Barnhart C., Ware, K. and Reinhardt, G. (1999). Multimodal express package delivery: a service network design application. *Transportation Science*, 33(2d), 391-407.

Kohl, N., Desrosiers, J., Madsen, O.B.G., Solomon, M.M. and Soumis, F. (1999). 2-Path cuts for the vehicle routing problem with time windows. *Transportation Science.* 33(1) 101–116.

Kontoravdis G, Bard JF (1995). A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7(2a), 10-23.

Kwon, O.K., Martland, C.D. and Sussman, J.M. (1998). Routing and scheduling temporal and heterogeneous freight car traffic on rail networks. *Transportation Research Part E*, 34 (2), 101–115.

Langevin A. and Soumis F. (1989). Design of multiple-vehicle delivery tours satisfying time constraints. *Transportation Research Part B: Methodological*, 23(2b), 123-38.

Lawley, M., Parmeshwaran, V., Richard, JP., Turkcan, A., Dalal, M. and Ramcharan, D. (2008). A time-space scheduling model for optimizing recurring bulk railcar deliveries. *Transportation Research Part B*, 42(5), 438-454.

Lee Y.H., Kim, J.I., Kang, K.H. and Kim, K.H. (2008). A heuristic for vehicle fleet mix problem using tabu search and set partitioning. *Journal of the Operational Research Society*, 59(6), 833-841.

Lee Y. and Sherali, H.D. (1994). Unrelated machine scheduling with time-window and machine downtime constraints: an application to a naval battle-group problem, *Annals of Operations Research*, 50(1), 339-365.

Li, H. and Lim. A. (2001). A metaheuristic for the pickup and delivery problem with time windows. *ICTAI-200*1, 13*th IEEE Conf. Tools Artificial Intelligence*, Dallas, TX, 160-170.

Mitrović-Minić, S. and Laporte, G. (2006). The pickup and delivery problem with time windows and transshipment. *INFOR*, 44(2c), 217-227.

Mitrović-Minić, S. (1998). Pickup and delivery problem with time windows: a survey. *SFU CMPT TR* 1998, 12(2a), 1-43. ftp://ftp.fas.sfu.ca/pub/cs/TR/1998/CMPT1998-12.pdf.

Nanry, W. P. and Barnes, J. W. (2000). Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34(2b), 107-121.

Nemhauser, G.L. and Wosley, L.A. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, New York.

Osman, I.H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4), 421-451.

Parragh, S.N. (2011). Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C*, 19(5), 912-930.

Parragh, S.N., Doerner, K.F. and Hartl, R.F. (2008). A survey on pickup and delivery problems. Part II: transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58(2), 81-117.

Parragh, S.N., Doerner, K.F. and Hartl, R.F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6), 1129-1138.

Prins, C. (2002). Efficient heuristics for the heterogeneous fleet multitrip VRP with application to a large-scale real case. *Journal of Mathematical Modelling and Algorithms*, 1(2), 135-150.

Qu, Y. and Bard, J.F. (2012). A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. *Computers & Operations Research*, 39(10), 2439-2456.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(2d), 455-472.

Ropke, S., Cordeau, JF. and Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(2d), 258-272.

Ropke, S. and Cordeau, JF. (2009). Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(2c), 267-286.

Ruland, K. S. and Rodin, E. Y. (1997). The pickup and delivery problem: faces and branch-and-cut algorithm. *International Journal of Computer Mathematics*, 33(2l), 1-13.

Savelsbergh, M., and Sol, M. (1995). The general pickup and delivery problem. *Transport Science*, 29(2a), 17-29.

Shang, J. S., and Cuff, C. K. (1996). Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering*, 30(2d), 631-645.

Shao, Y. (2011). Physical therapist network routing. Dissertation, Graduate Program in Operations Research & Industrial Engineering, The University of Texas, Austin.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical report, Department of Computer Science, University of Strathclyde, Scotland*.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. *Lecture Notes in Computer Science 1520*; *Principles and Practice of Constraint Programming*, M. Maher and JF. Puget (Eds.), Springler-Verlag, Berlin, 417-431.

Sherali, H.D. and Tuncbilek, C.H. (1997). Static and dynamic time–space strategic models and algorithms for multilevel railcar fleet management. *Management Science*, 43 (2), 235–250.

Sol M. and Savelsbergh M.W.P. (1994). A branch-and-price algorithm for the pickup and delivery problem with time windows. *Memorandum COSOR* 94-22, Department of Mathematics and Computing Science, Eindoven University of Technology, Eindoven, The Netherlands.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2b), 254-265.

Spoorendonk, S. and Desaulniers, G. (2010). Clique inequalities applied to the vehicle routing problem with time windows. *INFOR*, 48(1), 53-67

Steinzen, I., Gintner, V., Suhl, L. and Kliewer, N. (2010). A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3), 367-382.

Thangiah, S. R., Fergany, A. and Awan, S. (2007). Real-time split-delivery pickup and delivery time window problems with transfers. *Central European Journal of Operations Research*, 15(2d), 329-349.

Thengvall, B.G., Bard, J.F. and Yu, G. (2000). Balancing user preferences for aircraft schedule recovery during irregular operations. *IIE Transaction*, 32(3), 181-193.

Toth, P., and Vigo, D. (editors) (2001). The Vehicle Routing Problem, *SIAM Monographs on Discrete Mathematics and Applications*, Philadelphia.

van den Akker, J.M., Hurkens, C.A.J. and Savelsbergh, M.W.P. (2000). Time-indexed formulations for machine scheduling problems: column generation. *INFORMS Journal on Computing*, 12(2), 111-124.

Wosley L.A. (1998). *Integer Programming*. John Wiley & Sons, New York.

Xu H., Chen Z. L., Rajagopal S. and Arunapuram S. (2003). Solving a practical pickup and delivery problem. *Transport Science*, 37(2c), 347-364.

Yan, S., Chen, S. C., and Chen, C. H. (2006). Air cargo fleet routing and timetable setting with multiple on-time demands. *Transportation Research Part E: Logistics and Transportation Review*, 42(2e), 409-430.